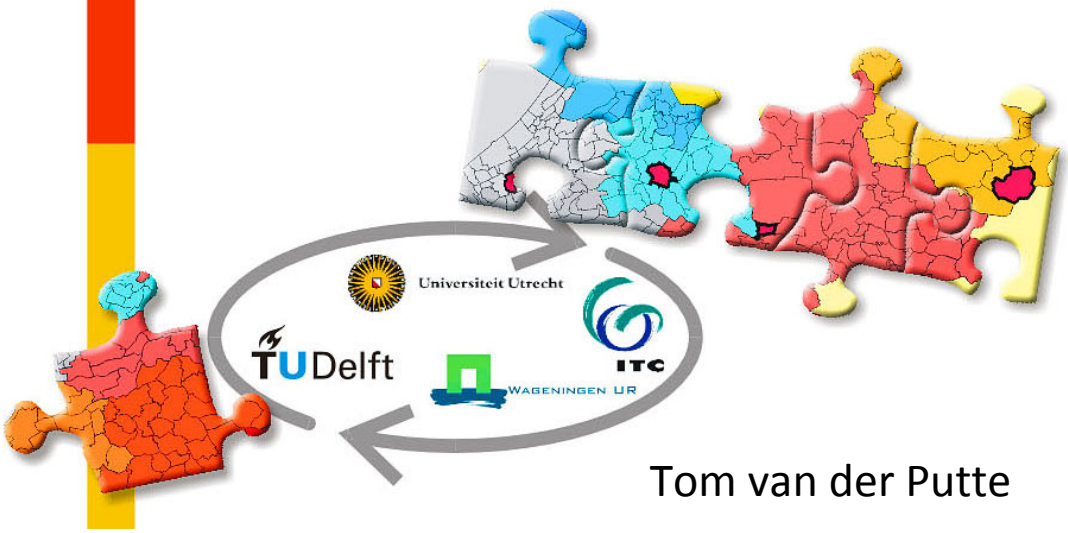
GIMA

Geographical Information Management and Applications

# *Using the discrete 3D Voronoi diagram for the modelling of 3D continuous information in geosciences*




Tom van der Putte

# *Using the discrete 3D Voronoi diagram for the modelling of 3D continuous information in geosciences*

Master Thesis
Geographical Information Management and Applications

**Author:**
Tom van der Putte,
Utrecht University

**Supervisor:**
Dr. H. Ledoux
Delft University of Technology

**Professor:**
Prof. dr. ir. P.J.M. van Oosterom,
Delft University of Technology

**Reviewer:**
Dr. D. Karssenberg,
Utrecht University

## ACKNOWLEDGEMENTS

First of all, I would like to thank my girlfriend, and biggest support: Natanja. I am convinced that this thesis would not have existed if it weren't for her ability to run the practical side of both her and my everyday life while I was locked up in the world of Voronoi diagrams, behind my computer screen.

Also, I would like to thank my parents for their support and all the opportunities they have given me up to now.

I would also like to thank Hugo for presenting the idea of the discrete 3D Voronoi diagram, and for helping me out when through the edges, I could no longer see the Voronoi diagram....

# Abstract

In this thesis the use of the discrete 3D Voronoi diagram in modelling continuous fields in geosciences will be assessed.

The Voronoi diagram is a structure that divides a certain space into cells, called Voronoi cells. These cells are created so that every location within each cell is closest to the corresponding data point, or seed, of that cell. Every location in that cell is assigned the value of the corresponding seed.

Throughout history, the Voronoi diagram has been widely used. In general, the Voronoi diagram is used in three spatial operations:

- Using the Voronoi diagram to determine spatial distribution and neighbourhood relations between points in a dataset.
- Using the Voronoi diagram to determine the area of influence of points in a dataset.
- Using the Voronoi diagram as a basis for the natural neighbour interpolation method.

In geosciences, data is sampled in many different ways. Sometimes by means of a regular grid for instance, but measuring continuous fields is often done in an anisotropically distributed pattern. Most often this is due to the way in which samples are collected. The Voronoi diagram can be a good means of visualizing and determining the distribution of data points, and therefore it can be a useful tool when working with anisotropically distributed data.

The exact 3D Voronoi diagram is a data model in vector format that has been investigated for quite some years now, often in different areas of sciences. The properties, the advantages and the disadvantages of this data model have been documented. The discrete 3D Voronoi diagram is a data structure in raster format, and, although it has been investigated, it has not been properly documented with respect to its properties, advantages and disadvantages. In this thesis these properties are described, especially in light of the modelling of continuous field data in the realm of geosciences, to perform for instance the natural neighbour interpolation, visualize the data distribution and other spatial operations.

To do so, first a new algorithm has been devised, based on literature reviews of different articles on the 3D discrete Voronoi diagram. The new algorithm presented in this thesis has been implemented through the Python programming language. In order to use the discrete 3D Voronoi diagram in combination with geo-scientific, continuous data, a GIS that handles 3D raster data was identified, namely GRASS, and the possibilities and functionalities of GRASS with respect to the discrete 3D Voronoi diagram were investigated. Some functionality, such as resampling and natural neighbour interpolation, that was not found in the GIS but that is considered necessary, has also been implemented.

By using test data, the entire process has been tested, from creating a discrete 3D Voronoi diagram, through interpolation it, to visualizing and analyzing it. The outcome shows that the discrete 3D Voronoi

diagram is a proper tool for handling and analyzing 3D continuous field data. It enables user to go from a dataset of 3D points to a 3D continuous field, in the environment of a GIS, which allows for even further spatial analyses and operations.

One of the advantages of the implemented algorithm is the effectiveness in adding, moving and removing points. This makes it very suitable for dynamic modelling, as well as natural neighbour interpolation. The algorithm also shows other interesting future possibilities, such as creating 3D generalized Voronoi diagrams. Some of these advantages and future uses are currently difficult or not yet possible to implement in an exact environment.

Besides the possibilities of the presented algorithm, practical implementation of the algorithm is not yet feasible, due to calculation-time related issues. However, it is expected that if a compiled programming language such as C++ is used, as opposed to the interpreted language Python, the efficiency of the algorithm will increase.

# TABLE OF CONTENTS

# 1. INTRODUCTION

In this thesis, the use of the discrete 3D Voronoi diagram in combination with continuous field data in geosciences will be evaluated. To introduce the main subjects that this thesis will cover, first the general concept of the Voronoi diagram will be presented, after which two forms of the VD will be introduced; exact and discrete. After these data structures have been explained, the main problem of this MSc research will be presented, which will finally be formed into the main research objective.

## 1.1. THE HISTORY AND USES OF THE VORONOI DIAGRAM

The concept and definition of the Voronoi diagram are quite straightforward. The Voronoi diagram is the result of a simple interpolation method. The definition of the Voronoi diagram is: A space containing points (seeds) that is divided into cells in such a way that every location within each cell is closest to the seed that lies within that cell compared to all other seeds. The boundaries of the cells represent the locations that are equidistant from two or more seeds (see Figure 1.1).

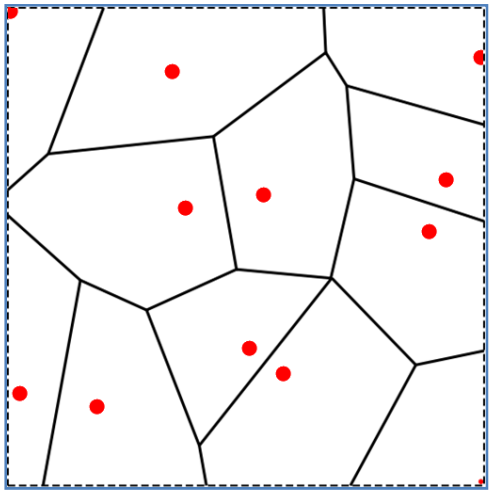


**Figure** 1.1 The Voronoi diagram. The red dots are the seeds, and the solid black lines the boundaries of the Voronoi cells.

The Voronoi diagram has been used in different fields of science since many years. According to Okabe, Boots et al. (1992), Descartes published several examples of a Voronoi diagram in his book *Le monde de Mr Descartes, ou Le Traité de la Lumiere* (Descartes, 1644). An example from this book can be seen in Figure 1.2. It shows the way in which objects are distributed in the Solar System. Note the circular shapes that are drawn within the different cells; these shapes will be considered in more detail when explaining the creation of the Voronoi diagram in Chapter 4. Both Dirichlet (1850) and Voronoi (1908) were one of the firsts to publish work that concerned the Voronoi diagram of which the publication is undisputed. This is the reason that the Voronoi diagram is also called the Dirichlet tessellation.

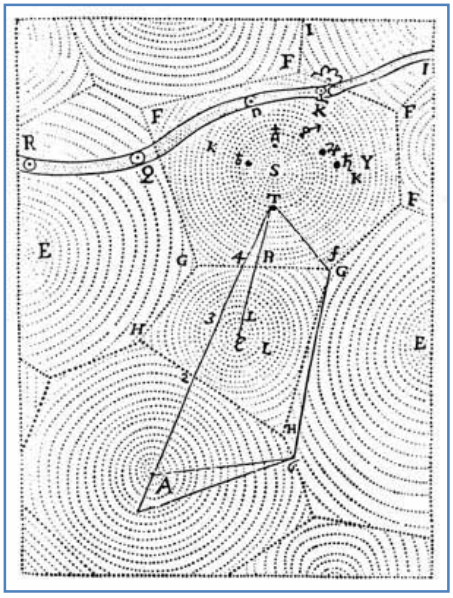


**Figure** 1.2 Representation of the solar system. *S* represents the sun. All the Voronoi cells represent heavens. From Descartes (1644).

One of the abilities that the Voronoi diagram displays is that of modelling features that are present in nature on very different scales. It can be used to model crystal lattices and other molecular structures in chemical sciences such as crystallography (Wigner and Seitz, 1933) and metallurgy (Spitzig, Kelly et al., 1985). But also in animal ecology and plant ecology the Voronoi diagram is used (Mead, 1967), as well as in astronomy and countless other fields.

One thing that the studies in which the Voronoi diagram is used have in common is the nature of the study. The aim is always to find a relation between a (series of) point(s) on the one hand, and either the area of influence around such a point, or the location with respect to its neighbours on the other hand (neighbourhood operations).

Another field in which the Voronoi diagram is often used, and which is of particular interest to me is the field of geosciences. Numerous applications of the Voronoi diagram within the realm of geosciences can be given (Tsai, 1993; Delerue, Perrier et al., 1999; Courrioux, Nullans et al., 2001; Tucker, Lancaster et al., 2001; Zhang and Thurber, 2005), all of which are again concerned with the area of influence of neighbourhood operations in one way or another.

## 1.2. THE VORONOI DIAGRAM IN GIS

The evolution of the personal computer has been a very influential factor in the field of geosciences, since it allowed for the creation of GIS (Geographical information Systems). There are several definitions proposed for the term GIS, by several people, but they all come down to the fact that a GIS is a system that allows for the representation, analysis, manipulation and storage of spatial information.

In a GIS, data is usually handled in two different environments; raster and vector representations (Winter and Frank, 2000; Heywood, Cornelius et al., 2006; Ledoux, 2006). Both environments have their own characteristics, and the accompanying pros and cons. A small summary is presented on both data representations.

### 1.2.1. VECTOR REPRESENTATION

Vector representation is exact, in the sense that points, lines and planes can defined in mathematical expressions, usually of the first order. They have an exact position and size. Some characteristics of objects represented in vector space are

- They are exact.
- They need little storage space.
- Object oriented
- Topology can be added

Vector representation can also be seen as an object oriented view: the polygons, (poly)lines and points describe objects in a space (Gold and Edwards, 1992). A polygon for instance can represent the footprint of a building. Another polygon which is adjacent to this polygon might represent the garage. Using topology the relationships (such as adjacency) can be described.

So a Voronoi diagram represented in a vector environment is in essence nothing more than a set of mathematical expressions, representing points and lines and polygons (and polyhedral in 3D). Figure 1.1 is an example of a visualization of an exact VD. Why Figure 1.1 is actually not a vector image is explained in the next section.

### 1.2.2. RASTER REPRESENTATION

Raster space is a discretized space. Usually it is tessellated using regular square tessellations called pixels (in 3D these are called voxels). Objects represented in raster space are also discretized, and are therefore called discrete. Discrete objects are always an approximation of the exact model, in the sense that an exact point or line can often not precisely be represented in pixels. Take for instance a point. In vector space, this point is defined to have a certain position, for example in Cartesian space. This point can be zoomed into indefinitely, but it will not grow larger for the observer, since an exact point has no radius or size. In raster space however, a point can only be represented as one (or more) pixel(s), with a discrete size. When an observer would zoom at this point, he would see a pixel which would grow larger with higher zoom levels.

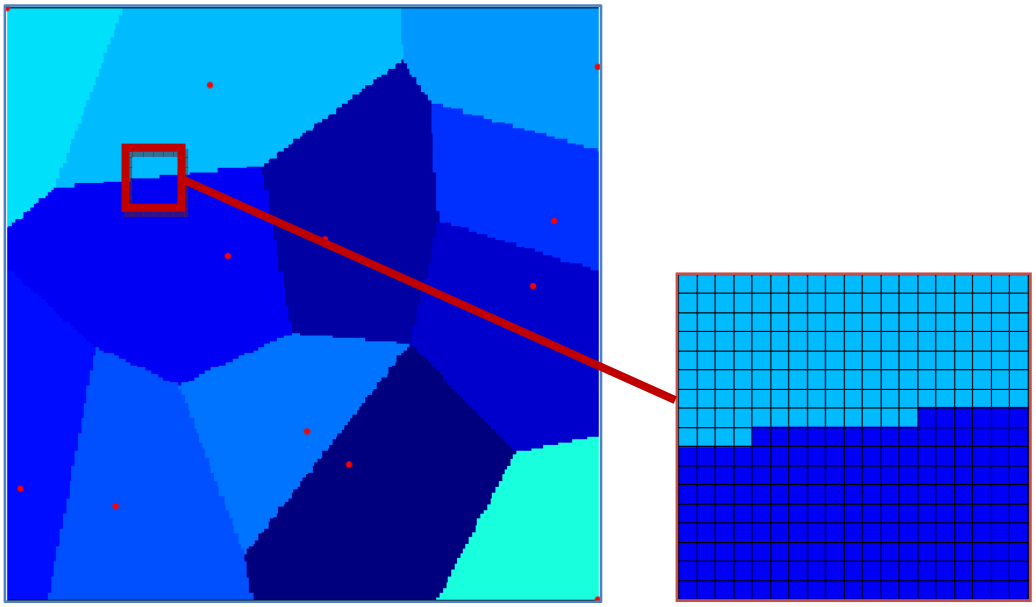
Some characteristics of the raster environment are:

- Rasters are widely used in practice.
- Rasters require relatively large amounts of storage space.
- Raster are simple structures
- Rasters are easily manipulated.

Also, the raster environment is does not really support an object oriented view, like the vector environment does. A raster describes tiles (pixels) which have an inherent topology, in other words, it is know which pixels are adjacent to others (Gold and Edwards, 1992). By making groups of pixels that have equivalent values, objects may be approximated. This concept is used in this thesis do represent a Voronoi partition, or cell; a set of adjacent pixels that have the same value.

Although an exact rational can be stored and used in a computer, displaying a vector as an images on a screen in its exact form is not possible. The reason for this is that computers work with discrete data: at the lowest level, objects in a PC are represented in 0's and 1's. Besides that, a computer screen is made-up out of pixels. Therefore, an image representing vector data is always a discrete image.

Figure 1.3 is a representation of the same Voronoi diagram of Figure 1.1, but now in raster space. The cells have a similar shape, but when zoomed in on an edge between two cells, the tessellations (pixels) become visible.



**(a)** **(b)**

**Figure** 1.3 **(a)** A Voronoi diagram in raster space and **(b)** zooming in on a small section of the Voronoi diagram shows the discrete nature of the image.

The exact Voronoi diagram has been used extensively in many fields of science. Some general fields have already been mentioned, and in the following section some examples of applications of the Voronoi diagram in the realm of geosciences will be presented. The discrete Voronoi diagram, however, has not been used so frequently.

### 1.2.3. THE VORONOI DIAGRAM IN GEOSCIENCES

From the previous sections, it is clear that the Voronoi diagram is a widely used method with numerous applications. Also, within the field of geosciences, the Voronoi diagram is a tool that is used frequently (Gold, 1989; Gold and Edwards, 1992; Okabe, Boots et al., 1994; Delerue, Perrier et al., 1999; Courrioux, Nullans et al., 2001; Zhang and Thurber, 2005). As already mentioned, when the Voronoi diagram is used, it is often to find the area of influence of a certain set of points, or to find neighbourhood relations between points in a dataset. This is illustrated by the article by Gold and Edwards (1992), in which they propose that the Voronoi diagram can be utilized as "*a general spatial method with applications in remote sensing, GIS and other spatial disciplines*". However, the Voronoi diagram can also be used in different ways.

The Voronoi diagram is a space divided into partitions, as explained in Section 1.1 (an exact equation is given in Section 3.1, Equation 1), and as such can be interpreted as an interpolated surface in 2D, in 3D as an interpolated volume; given a set of points a certain space, every location in that space is given a value based upon the position that the location has with respect to the data points. The interpolation method that returns a Voronoi diagram is called the nearest neighbour interpolation method. One could argue that it is a crude method, since it does not yield a continuous surface. However, it does give a good overview of for instance the distribution of data points, and the relation between points that at first glance might not be connected. This makes the Voronoi diagram very suitable to use in combination with anisotropic data. In geosciences, data is often collected in an anisotropic fashion. For instance, collecting temperature measurements in the earth's subsoil is done by drilling wells, and measuring the temperature in the wells at varying depths. This yields data that is distributed quite compactly in the vertical dimension, while the data points are distributed far apart in horizontal dimensions.

The fact that the Voronoi diagram is an interpolated surface does not mean that it cannot be interpolated any further. In fact, when performing natural neighbour interpolation, the Voronoi diagram is a prerequisite (Sibson, 1980; Sibson, 1981; Park, Linsen et al., 2006). The natural neighbour interpolation, also called Sibson interpolation, is an interpolation method that yields relatively accurate surfaces, but which is difficult to implement in exact environments (Perez and Traversoni, 1996; Ledoux, 2006). The natural neighbour interpolation will be discussed in detail in Section 4.6.2.

There is also another way to utilize the Voronoi diagram. In Ledoux (2006), the author proposes to use the exact 3D Voronoi diagram as the basic data model in which to represent trivariate data as an alternative to a raster environment which is used in almost all cases to represent trivariate data. According to Ledoux, this is done for three main reasons. The first and second reasons have already

been mentioned; it gives a clear definition of the neighbourhood relations between the data points, and it is possible to (re-) create the continuous field using natural neighbour interpolation. The third reason is that the Voronoi diagram enables several visualization operations, as well as several spatial analysis operations. This will be shown in detail in sections 2.2, 2.4 and Chapter 5. There are more advantages in using the Voronoi diagram, which will also become clear in further sections.

However, the exact Voronoi diagram is rather difficult to implement in a robust and efficient way (Sugihara and Inagaki, 1995). First of all, 3D geometrical computations are difficult in general, but the construction of 3D Voronoi diagram, and especially dynamically modifying the Voronoi diagram, are both very intricate procedures for which very few and difficult algorithms exists. More details on the difficulties concerning the creation of the exact 3D Voronoi diagram will be given in Section 3.1.1.

One interesting question arose from this research: what would happen if not the exact 3D Voronoi diagram is used, but the discrete 3D Voronoi diagram? Both raster and vector space have their own advantages and disadvantages, and it is likely that both representations of the 3D Voronoi diagram have different pros and cons. What advantages of the exact representation will it retain, what advantages of the raster environment will be gained, and what disadvantages will there be?

To see if this idea could lead to a valid research objective, some preliminary investigations were performed. These show that indeed the discrete 3D Voronoi diagram may have some interesting abilities: the use of raster based methods might be of use creating Voronoi diagrams for objects other than points (lines, areas) (Li, Chen et al., 1999; Dong, 2008; Zhao, Li et al., 2008). This might also be the case for generalized Voronoi diagrams, which are Voronoi diagrams that are generalized in seeds, assignment function and/or space (Drysdale and Lee, 1978; Okabe, Boots et al., 1994) (Section 5.2.2). Both these operations are difficult to implement in an exact environment.

The discrete 3D Voronoi diagram might also be of use in situation in which dynamic modelling is required. Dynamic modelling is used to (visually) modify or manipulate data in order to assess data and the distribution of data, and to see what effect possible changes in data might have. The main tool that provides such functionality would is a tool that allows the user to add, move and remove certain data points. Although this tool has been provided in the exact Voronoi diagram data structure proposed by Ledoux (2006), it is again difficult to implement, especially in certain degenerate cases. The discrete 3D Voronoi diagram might allow for easier implementation of these dynamic modelling tools.

The fact that the discrete 3D Voronoi diagram might be helpful in these situations is promising, however, this thesis is written from a GIS-perspective. This means that the solution must be practical and usable. Therefore it is important to know if the 3D Voronoi diagram can be used within, or in combination with a GIS, and how this can be achieved. These questions finally lead to the research objectives that are stated in the next section.

## 1.3. Research Objectives

### 1.3.1. Main research objective

Based on the description of the problem in the previous introduction, the main research objective that can be posed is as follows.

> ***To assess the use of the discrete 3D Voronoi diagram for the modelling of 3D continuous information in geosciences.***

### 1.3.2. Research questions and objectives

The main objective can be reached by answering the following research questions. The research questions are divided in sub-questions and corresponding objectives.

**A)** What are the main differences between the discrete and exact 3D Voronoi diagram?
- What are in theory the main differences between exact and discrete (3D) Voronoi diagram?

**B)** How can a discrete 3D Voronoi diagram be created?
- In what ways can a discrete 2D Voronoi diagram be created?
- How can an algorithm for the creation of a 2D Voronoi diagram be converted into one that can create a 3D Voronoi diagram?
- Create and implement an algorithm to create discrete 2D Voronoi diagrams.
- Convert the 2D algorithm in order to create discrete 3D Voronoi diagrams
- Adapt the algorithm to enable it to add, move and remove points.

**C)** Which software packages are suitable for use with the discrete 3D Voronoi diagram?
- Which software packages can handle 3D raster data files?
- Which of these are of use to the analysis of continuous fields in geosciences?
- What data formats are the software packages able to read/import?
- What functionality should be available within these software packages?
- Devise storage and conversion algorithms.
- Expand program with a storage/conversion routine to store or convert the model into usable raster format(s).
- Identify lacking functionality.
- Include lacking functionality into the design of the discrete 3D Voronoi diagram algorithm.

**D)** How does the exact 3D Voronoi diagram compare with the discrete 3D Voronoi diagram?
- What are the pros and cons of both
- How can the discrete 3D Voronoi diagram be utilized in GIS?
- What improvements can be made to the discrete 3D Voronoi diagram in the future?

### 1.3.3. SCOPE

The aim of this research is to design and implement an algorithm that can produce a discrete 3D Voronoi diagram, in a robust manner, as well as add, remove and move points within the discrete 3D Voronoi diagram. This is necessary to provide a structure that can be dynamically modified which is an important functionality in current GIS. The scope of this research also comprises the assessment of the use of the discrete 3D Voronoi diagram in combination with current GIS. This means that different functionalities (among which are analytical and visualization functionalities) will be tested. If certain functionalities are not present, but are deemed necessary for this research, development of such functionalities will be attempted. The discussion of future possibilities of the discrete 3D Voronoi diagram also fall within the scope of this research.

It is not within the scope of this research project to create the most efficient algorithm possible. This is due to the fact that this is not a computer science thesis, and in this case, conceptual simplicity is a factor that has a major influence. However, as far as time will allow, efficiency will be pursued as a sub-objective in order to make the algorithm more usable.

Although storing 3D raster in different file formats will be implemented to allow for a degree of interoperability if necessary, storing the 3D raster as efficient as possible (through compression) is not within the scope of this thesis.

Also, in light of calculation time, parallelization (executing the program on multiple CPU cores) might decrease the calculation time of the algorithm. However, this does not fall within the scope of this research.

The aim of this research is to create an algorithm that can process 3D points that contain an attribute value (x,y,z,a), which is the most common form of sampling in geosciences. Creating discrete 3D Voronoi diagrams of other objects, such as lines, planes and polygons, is not within the direct scope of this project. However, the possibilities of the algorithm for future adaptations are also important. Therefore, this might indirectly have consequences for the algorithm that is created. This point will be discussed in Chapter 5.

# 2. STATE OF THE ART IN 3D GIS

In this chapter the start of the art in GIS with respect to 3D rasters will be investigated and presented. In order to find a suitable GIS, a set of prioritized criteria (such as means of visualization and analytical functionality) is presented to describe what a GIS must be able to handle or execute. Based on these criteria, GIS's are evaluated and one GIS is selected to be used during the research. Functionalities that are not available in the selected GIS but are deemed necessary for this research is identified.

## 2.1. INTRODUCTION

Although the algorithm that is designed for this project has not been presented yet, the resulting output will consist of a discrete 3D Voronoi diagram, and will be in a 3D raster format. It is important to note that this research is concerned with true 3D data (or volumetric data) that consists of x,y,z coordinates and one or more additional attribute value. This is contrary to the so-called 2.5D data, where data points consist of only x and y coordinates, and an attribute that denotes the z-value (elevation).

The discrete 3D Voronoi diagram can in most circumstances not be considered to be an end-product. Instead the power of the Voronoi diagram lies in enabling the user to perform different operations and analyses on the dataset. The ideal environment to perform these operations and analyses is a GIS, which is able to handle georeferenced data of different origins and formats, visualize and analyze it.

The GIS selected for this project is GRASS 6.3 (GRASS Development Team, 2008). GRASS is an open source and freeware GIS, that supports 3D raster files. For visualization purposes, MayaVi2 (Ramachandran and Varoquaux, 2008) is used. In this chapter, several GIS are presented, and it is shown that, based on several criteria, the combination of GRASS 6.3 and MayaVi2 is the most suitable for this research project.

## 2.2. CRITERIA

Before selecting suitable GIS, a set of criteria must be determined to which the GIS must adhere to. Three different types of criteria are distinguished: standard, analytical and visualization. These criteria are shown and described in Table 2.1.

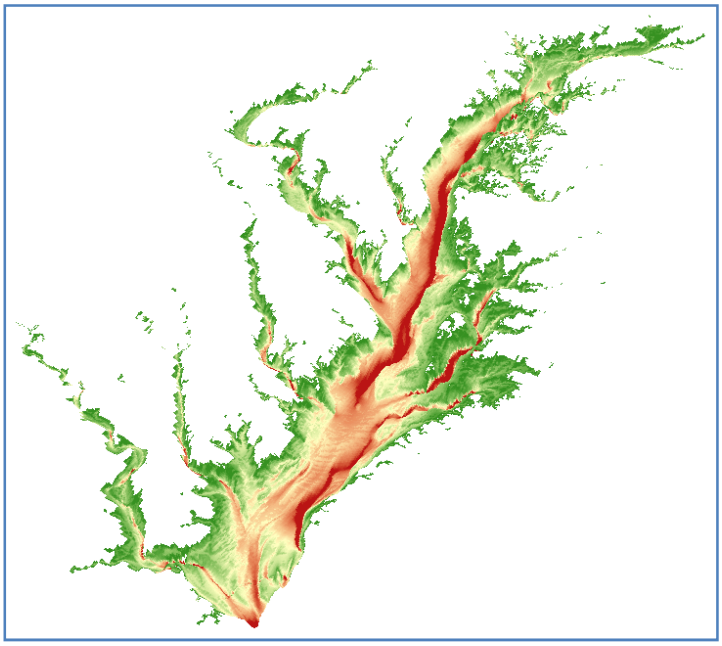
| Nr. | Criterion | Type of functionality | Priority | Description |
|---|---|---|---|---|
| 1 | Import 3D raster | Standard | Critical | The GIS must be able to read 3D raster files. Not the standard 2.5D representation, which has been around since the 1980's (Bak and Mill, 1989). |
| 2 | Map Algebra | Analytical | High | Map algebra is the collection of tools that enable the user to perform calculations with (3D) raster files, as described in Tomlin (1990). |
| 3 | General 3D raster statistics | Analytical | High | Provides statistics such as dimensions, volume/area calculations etc. |
| 4 | Resampling 3D raster | Analytical | Medium | Resampling is a standard basic operation in 2D GIS, and often used. |
| 5 | Interpolating 3D point sets | Analytical | Medium | Interpolating 3D points can be useful to combine the results of these interpolations with the 3D raster |
| 6 | Slicing | Visualization | High | Slicing a 3D raster will enable the user to view a slice (plane in any direction) of the 3D raster in 2D |
| 7 | Isosurface | Visualization | High | Isosurface is a surface that connects points with the same value. It is the 3D equivalent of the 2D isoline. |
| 8 | Volume rendering | Visualization | Low | A volume rendering of a 3D raster is a way to visualize the entire grid on a 2D plane (the screen), giving a semi – transparent image of the 3D raster. |

**Table** 2.1. Criteria that the tested GIS must adhere to.

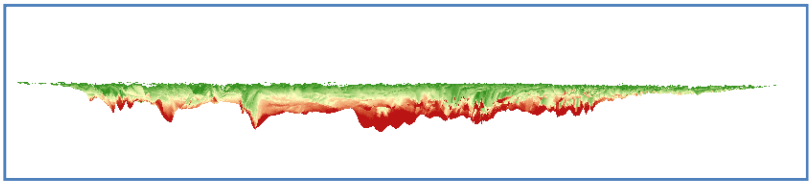
Some remarks on this table are in place. First of all the criteria described here have been assigned a priority. Criterion 1, the ability to import 3D rasters is critical; without this, the 3D Voronoi diagram cannot be utilized. Therefore, GIS that do not adhere to this criterion will not be selected.

Also, as mentioned in Section 2.1, the so-called 2.5D visualization is not the same as 3D. Data that has an x and y coordinate, and an attribute value can sometimes be visualized in 2.5D. When this attribute represents for instance elevation, most GIS can project the data such that a seemingly 3D image is shown. However, this is nothing more than a surface that is elevated to the attribute value at each associate (x,y) coordinate. Figure 2.1 shows a 2.5 D surface. In this case, it is a DTM (Digital Terrain Model) that shows the elevation in a certain part of Slovakia.

Data that is truly 3D however, has x, y and z values and one (or more) attribute value(s) (Tse and Gold, 2004), and is also called volumetric data.



**(a)**



**(b)**

**Figure** 2.1. The DTM (bathymetry) of the Chesapeake Bay estuary is an example of a 2.5D surface, shown from two different angles: **(a)** oblique top view and **(b)** parallel to horizontal plane.

### 2.2.1. VISUALIZATION CRITERIA

The visualization criteria are based upon three methods that are frequently used for displaying discrete 3D information. The first method is slicing, which is basically the visualization of the intersection between a plane and a 3D raster. The flat surface which is obtained is called a slice. Slicing can give valuable insights in the internal structure of a 3D raster, especially when multiple adjacent slices are animated. Isosurfacing is another technique, described in Kaufman (1994) . Isosurfacing is a method which extracts a surface from a 3D raster. All points on the surface have the same attribute value. Since 3D rasters consists of discrete elements (voxels) it is not simply a case of selecting voxels that have the same value, but intermediate values have to be inferred from the voxels in the grid. There are several methods for this (Lorensen and Cline, 1987; Van Gelder and Wilhelms, 1994; Carr, Snoeyink et al., 2003;

Newman and Yi, 2006), but the technical aspect of these is not within the scope of this thesis. More important is the result. Figure 2.2 shows the same 2.5D DTM as in Figure 2.1, but now simplified (only the main estuary is modelled). Combined with this 2.5D DTM is a 3D raster, of which several isosurfaces are extracted. These isosurfaces represent different concentrations of nitrogen in the water.

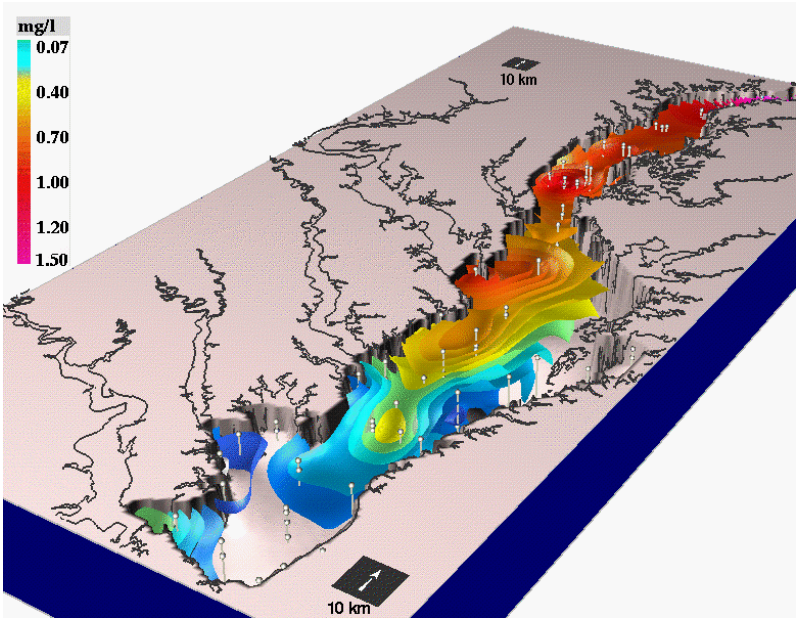



**Figure** 2.2. A 2.5D DTM combined with 3D data.
The 3D data is visualized by the blue and orange isosurfaces.
(From Mitasova, Mitas et al (1995))

The clarity of the resulting image and the ease of interpreting it, means that this way of visualization has a high priority.

Volume rendering is another method of visualizing 3D raster data (Kaufman, 1996). In this case, it is not only a part of the dataset that is visualized, as in slicing, or isosurfacing, but it is a way in which to show the entire 3D raster in once. This is done by assigning each voxel a certain colour and opacity (transparency). Then the projection of the 3D raster onto the screen is calculated, which is the image the user is shown. The advantage of this is that all information can be seen at once, which enables the user to place certain trends or object in context with the surrounding data. The major disadvantage of using it with the discrete 3D Voronoi diagram is that the raster is always in the form of a cube or a cuboid in which every voxel has a value. Trying to represent all those Voronoi cells in one 2D projection will not yield an image that can be readily interpreted by the user; therefore, this criterion has a low priority.

The power of the mentioned techniques for visualizing 3D raster data can increased dramatically by adding extra functionality such as perspective, rotation and motion. This way, the spatial interrelation that object or parts of objects have becomes much clearer.

## 2.3. EVALUATED GIS / SOFTWARE SUITES

For the purpose of this research, I have evaluated a few well-known GIS packages. These include: MapInfo Professional 9.0, ArcGIS 9.2, GRASS 6.3, Manifold Professional 8.00 and PCRaster 3D. An extensive review based on the criteria described in Table 2.1 would be given here but there was only one program that managed to fulfil the first critical criteria: *handling 3D raster data*. This package is GRASS 6.3. ArcGIS claims to support 3D raster files, and the software can indeed import NetCDF files (a 3D/4D raster format), but to the best of my knowledge, it will slice the data and present only a 1 voxel thick slice of the 3D raster (ESRI, 2009). PCRaster 3D was evaluated based on the paper by Karssenberg and De Jong, 2005. PCRaster 3D is a program focussed primarily on dynamic environmental modelling. It can handle 3D raster data, but the voxels are not fixed in size in the z-direction. Also, the software is in the prototype stage, so it is not available on the market as of yet. These reasons make it not suitable for use with the discrete 3D Voronoi diagram.

In fact, to the best of my knowledge, GRASS is the only GIS that meets the first criteria. Table 2.1 shows how GRASS handles the other criteria. The functionality presented in this table was based on a review of the online product specification and support documents (GRASS GIS, 2008).

| Nr. | Criterion | GRASS 6.3 |
|---|---|---|
| 1 | Import 3D raster | **Yes** |
| 2 | Map Algebra | **Yes** |
| 3 | General 3D raster statistics | **Yes** |
| 4 | Resampling 3D raster | **No** |
| 5 | Interpolating 3D point sets | **Some** (Bicubic or bilinear spline (GRASS GIS, 2007)) |
| 6 | Slicing | **Yes** |
| 7 | Isosurface | **Yes** |
| 8 | Volume rendering | **No** |

**Table** 2.2. Criteria that GRASS complies with

Based on this list of functionality, one could say that, besides being the only candidate, GRASS would qualify as a relatively good candidate to make use of the discrete 3D Voronoi diagram.

Although GRASS may be the only GIS available that can handle 3D rasters, there is specialized software available, predominantly in the geology, geophysics, and petrology area that is capable of handling 3D raster data. Software suites such as Petrel (Schlumberger, 2009), JewelSuite (JewelSuite, 2009) and EarthVision (Dynamic Graphics Inc., 2008) are all capable of handling 3D raster and 3D vector data.. But since none really qualify as an off-the-shelf GIS, and because licences for these packages cost approximately 10-100 time more than licenses for the GIS packages mentioned, these will not be discussed further in this thesis.

## 2.4. GRASS 6.3

Based on the results presented above, it is clear that there is one GIS that can be deemed suitable for handling the discrete 3D Voronoi diagram, namely GRASS 6.3. To further evaluate the usefulness of this package, Grass has been installed. The package has been installed on Windows XP Professional, Windows Vista Ultimate and Linux Ubuntu 8.10. Ultimately the Linux installation has been chosen, as it turned out to be the most stable.

In the next section, a small review of the product is given.

### 2.4.1. USER REVIEW OF GRASS

Grass is a GIS that can be controlled with a GUI (Graphical User Interface) as well as a with a command-line interface. The program consists of several modules, one of which is the NVIZ module (GRASS GIS, 2008). NVIZ is a visualization module that enables users to import and visualize 2.5D and 3D raster files, as well as 2D and 3D vector objects (points, lines, polygons and polyhedra). In Figure 2.3 the interface of the NVIZ module is shown, with a 3D raster file that is visualized by two isosurfaces.

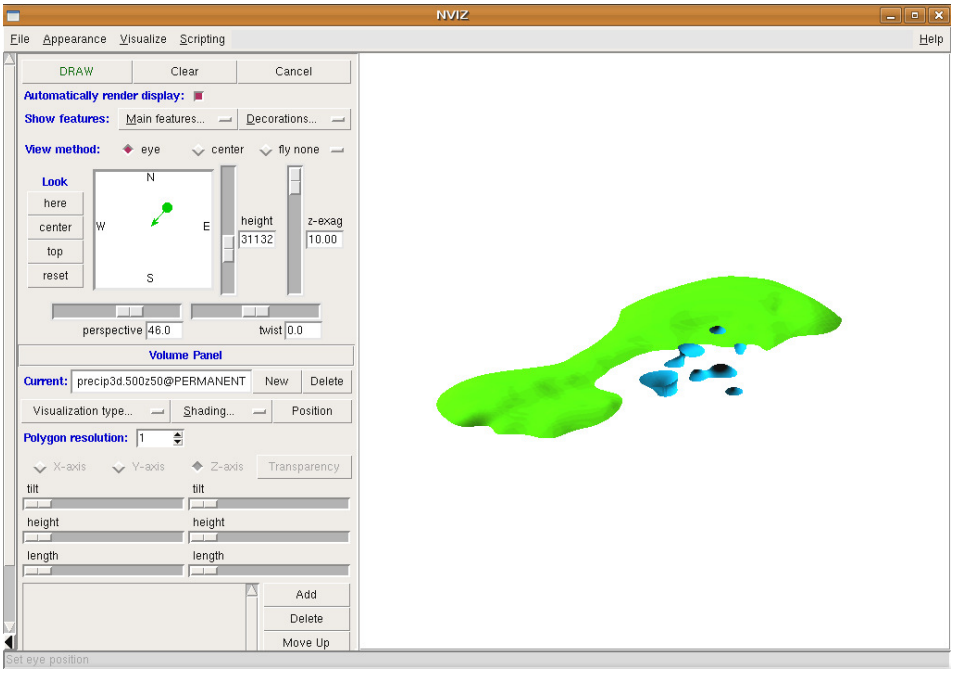



**Figure** 2.3. NVIZ visualization module showing 2 isosurfaces from a 3D raster

Almost all visualization functions described in Table 2.1 are available; isosurfacing (such as in Figure 2.2 and 2.3), as slicing and combining 2D and 3D rasters. The only function that is missing is volume rendering, but as explained in Section 2.2, this has not really a high priority. At first glance, the NIVIZ module seems to meet all requirements. But it is rather buggy, and not user-friendly.

### 2.4.2. DIFFERENT WAY OF VISUALISING: MAYAVI2

Because of the practical limitations of the NVIZ module, a different means of visualizing was sought. The requirements of this package were the following:

- Able to import and handle 3D raster files.
- Able to visualize 3D raster files in the ways described in Table 2.1.
- High degree of user-friendliness.

User-friendliness is a subjective term, but in this case, it simply means that it should be easy to work with, and not have a very steep learning curve.

There are plenty of packages which specialize in the visualization of 3D environments, such as various CAD programs, and other scientific geo-visualization software. After testing several programs, there was one program that stood out, and which was also free for academic uses. It is called MayaVi2 (Ramachandran and Varoquaux, 2008), and acts as an interface to the Visualisation Toolkit (VTK, 2009) which is a software library for visualizing 3D images. It can handle various formats of 3D files, in both vector and raster format. It adheres to the three criteria set above, not in the least to the third: user

friendliness; it is very intuitive, and comes with a simple menu structure. It is also quite easy to extend MayaVi2's functionality, since it has an interface for the Python programming language. MayaVi2 will be used as the main visualization program in this project.

Although MayaVi2 is selected as a suitable visualization tool, one major disadvantage is the fact that it cannot project georeferenced files, meaning that two 3D rasters in different projections cannot easily be visualized at the same time using MayaVi2.

## 2.5. STORAGE

In order to take full advantage of the power of the 3D Voronoi diagram, the visualization criteria have been separated from the analytical functionality. For the GIS analyses GRASS will be used, for visualization purposes, MayaVi2 will be used. Although this is a solution that meets all the criteria from Table 2.1, and uses user-friendly software, it also poses a new problem: after the discrete 3D Voronoi diagram has been analyzed in GRASS, it is necessary to transfer the data to MayaVi2 to visualize the outcome. This can only be done by exporting the data to a file from the GRASS program, and then importing it into MayaVi2. The methods of importing and exporting data in both MayaVi2 and GRASS have been identified, and are illustrated in Figure 2.4.

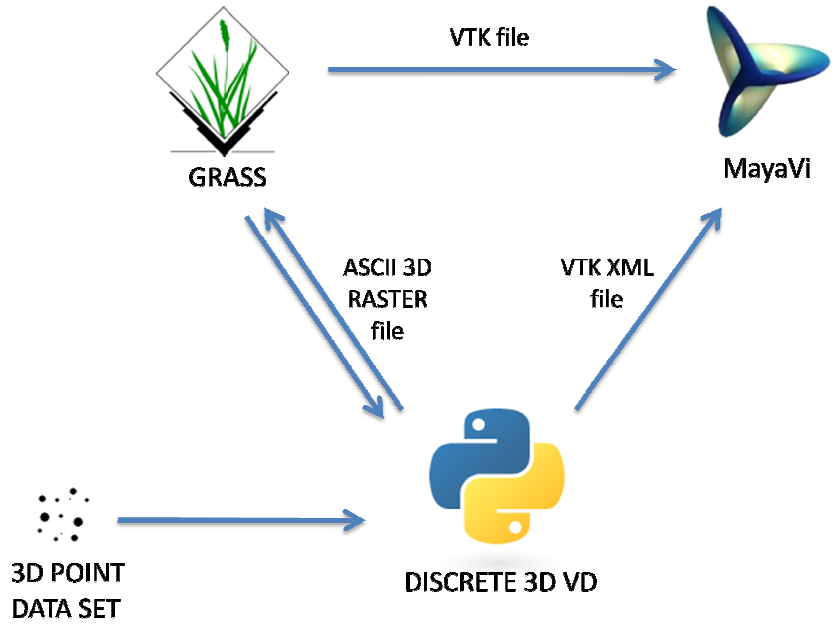



**Figure** 2.4. Schematics showing the different software packages and the accompanying data formats

By using these file formats all available functionalities in every program can be utilized. The internal structure of these files is presented in Appendix I. Note that it is not possible to export data from MayaVi2; it is used for visualization purposes only.

## 2.6. LACKING FUNCTIONALITY

In Table 2.2 shows that the resampling functionality and the volume rendering functionality are not available in GRASS, but since the visualization was assigned to MayaVi2, the only functionality that is not available is the resampling function. Because resampling is a frequently used tool, for instance for the rotation and resizing of rasters, it will be developed for this research and added to the general algorithm.

Although not specifically mentioned in the list of necessary functionality, but rather under the heading 'interpolation of 3D points', another analytical functionality will also be added. It is the natural neighbour interpolation method. This is an interpolation method that uses the Voronoi diagram as a basis. It delivers smooth results, but can be difficult and time consuming to implement when the exact Voronoi diagram is used (Perez and Traversoni, 1996). But initial reviews of articles indicate that this interpolation method might be easier to implement in raster environment (Park, Linsen et al., 2006) this might increase the effectiveness and efficiency of the natural neighbour interpolation. To investigate this, the natural neighbour interpolation will also be added to the general algorithm. It the implementation is successful, the usefulness of the discrete 3D Voronoi diagram will be improved greatly.

The implementation of both added functions will be presented in Section 4.6.

## 2.7. SUMMARY

In this chapter the state of the art in 3D (raster) GIS has been evaluated. Most GIS support 2.5D raster data and deliver some functionality to analyze these data. However, to measure the support for true 3D raster data, a set of prioritized criteria have been developed, and the selected GIS were assessed based on these criteria. There is only one GIS that meets the most important criteria; support of 3D raster data. This GIS is called GRASS, and is an open source freeware package.

Furthermore, the functionality that GRASS offers for the analysis, manipulation, visualization and storage has been inventoried and, combined with a created list of functionality that is considered necessary for handling 3D raster data, a set of functionalities that are not currently available in grass is identified. Part of the objective of this thesis is to implement these 'missing functionalities'. How this is done is explained in Chapter 4.

The visualization of 3D raster (one of the necessary functionalities) has also been discussed. The GRASS 3D visualization module does present the user with the required functions, however, in a user–unfriendly way and it seems to contain some bugs. Therefore a different visualization method was found. A program called MayaVi2 delivers almost all the necessary visualization options, in a very user-friendly GUI. The only disadvantage is that it cannot georeference or transform geographic data. Combining two rasters in different projections will therefore be difficult.

# 3. STATE OF THE ART IN DISCRETE 3D VORONOI DIAGRAMS

Although the title of this section refers to the discrete 3D Voronoi diagram, the exact Voronoi diagram will be explained in more detail at first in Section 3.1: how it is created and what is used for. The explanation of the exact Voronoi diagram will take place in 2D, and is later expanded to 3D. This will provide the necessary context for discussing the discrete Voronoi diagram.

After revisiting the exact version of the 3D Voronoi diagram, the discrete 3D Voronoi diagram will be presented in more detail (Section 3.2). First of all, the concept of the discrete 3D Voronoi diagram will be explained more thoroughly, as well as the concept of raster space and the different metrics which are possible within raster space. In Section 3.3 the different ways in which a discrete Voronoi diagram (2D and 3D) can be created will be presented, and the advantages and disadvantages of these methods will be discussed in terms of efficiency, simplicity and the possibility or adapting the algorithm.

## 3.1. THE EXACT VORONOI DIAGRAM

As already mentioned in the introduction, the exact Voronoi diagram is modelled in vector space. In 2D vector space, points, lines and polygons are primitives. The primitives in higher dimensions (in the case of this thesis the third dimension) are built upon the lower dimensionality primitives, and for the third dimension, the polyhedron is added.

There are a lot of variations on the ordinary Voronoi diagram (Okabe, Boots et al., 1994), but when the Voronoi diagram is mentioned in this thesis, the ordinary Voronoi diagram is meant. If a non-ordinary Voronoi diagram is discussed, it will be expressed so explicitly.

One way to illustrate the concept of the Voronoi diagram is to think of it as follows:

Let a finite Euclidian space contain a certain number (*n)* of points (with $n > 1$), called seeds. Around each of these seeds a circle is drawn, with a radius of 0. These circles expand at the same rate with an infinitely small increment. They will keep expanding, but where two circles touch, a line or boundary is formed. The circles keep expanding until the circles cannot expand any further (Figure 3.1). The resulting graph is the Voronoi diagram for that set of points. When looking at Figure 1.2 by Descartes, in the introduction, it can be seen that he visualized a similar method of creation.

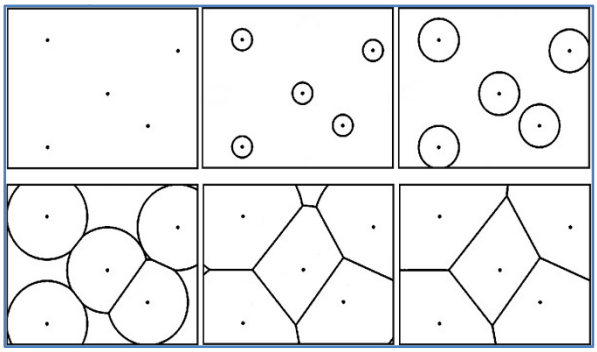


**Figure** 3.1 Visual representation of the formation of a Voronoi diagram

The polygons that make up the Voronoi diagram represent the areas in which each location is closest to the seed corresponding to that area. The mathematical equation of such a polygon $V(p_i)$ can be expressed as follows:

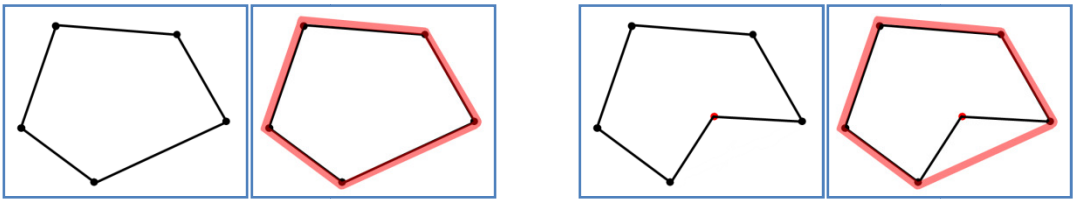
$$V(p_i) = \{p | d(p, p_i) \leq d(p, p_j), j \neq i, j = 1, \dots, n\}. \quad \text{Eq. 1}$$

**(From Okabe, Boots et al. (1992) (Okabe, Boots et al., 1992)**

Where:

- $p_1, \dots, p_n$ is a set of distinct seeds located in $\mathbb{R}^d$,
- $d(p, p_i)$ represents the Euclidean distance between location $p$ and seed $p_i$
- $V(p_i)$ represents the ordinary Voronoi polygon associated with seed $p_i$

These Voronoi polygons or cells are also called the *area of influence* polygons (Okabe, Boots et al., 1992). They are *convex*; which means that if an imaginative rubber band would be stretched around the vertices that make up the cell, the rubber band touch all vertices, whereas with a concave polygon, this is not the case. This is illustrated in Figure 3.2.



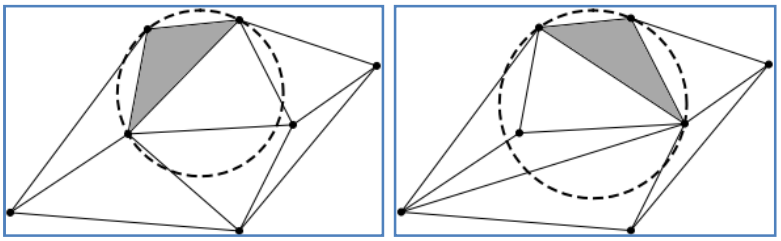
**(a)** **(b)** **(c)** **(d)**

**Figure** 3.2 **(a)** A convex polygon. **(b)** An imaginary rubber band (red line) stretched around the convex polygon touches every vertex. **(c)** A concave polygon. **(d)** An imaginary rubber band stretched around the concave polygon does not touch every vertex.

The Voronoi diagram has several interesting characteristics, such as the ability to illustrate spatial relations. (Okabe, Boots et al., 1994).

To create an exact Voronoi diagram within the bounds of a computer there are several options. In Okabe, Boots et al. (1992) numerous methods to create a Voronoi diagram in 2D are summarized and explained. Among those methods are the incremental method, the sweepline method, the walking method and the divide and conquer method. These all have their advantages and disadvantages and are suitable for different situations. Although these methods are quite interesting, they are beyond the scope of this thesis, and will therefore not be explained in detail. There is one method, however, which will be discussed in slightly more detail, because it is designed both to create a Voronoi diagram in 3D, and be able to insert, move and remove points at will, without having to recreate the entire structure. This method is based on the incremental method which starts out with a Voronoi diagram consisting of 2 or 3 seeds. By inserting new seeds one by one and updating Voronoi diagram after each insertion, this algorithm will eventually yield the entire Voronoi diagram of all seeds (Okabe, Boots et al., 1992). It is also based on the fact that the Voronoi diagram has a dual, called the Delaunay triangulation. To explain further, a small clarification on the Delaunay triangulation and the relationship between the Delaunay triangulation and the Voronoi diagram is in order.

The Delaunay triangulation can be defined in many ways, but one of the most simple definitions state that the Delaunay triangulation is a tessellation of space into triangles, where *n* points (the seeds in the Voronoi diagram, with: $3 \leq n < \infty$ ) represent the vertices of the edges, and where the triangles all have an empty circumcircle (Okabe, Boots et al., 1994). This means that within a circle that connects the three vertices of a triangle, there are no other points. Figure 3.3 a and b show an empty circumcircle, and non empty circumcircle respectively.



**(a)** **(b)**

**Figure** 3.3 **(a)** Empty circumcircle and valid Delaunay triangulation. **(b)** Non-empty circumcircle of the corresponding grey triangle, and invalid Delaunay triangulation.

The dual relation between the Voronoi diagram and Delaunay triangulation implies that if the Voronoi diagram is known, the Delaunay triangulation is derived from it and vice versa (in O(*n*) time). Duality in relation to graphs implies that the dual of a graph *G* has for each plane region (a Voronoi cell) in *G* a vertex, and for each edge in *G* an edge.

This is illustrated in Figure 3.4 where the Voronoi diagram is shown in solid lines, and the Delaunay triangulation in dotted lines. In each Voronoi cell, a vertex of the Delaunay triangulation is located. Also, each edge in the Voronoi diagram has an edge of the Delaunay triangulation. Also, a Delaunay triangle is dual to a Voronoi vertex. In fact each Voronoi vertex is located at the centre of the empty circumcircle that defines each Delaunay triangle (orange circle and vertex). For illustrational purposes, some edges are colour-coded; edges in both structures with the same colour are each other's duals.

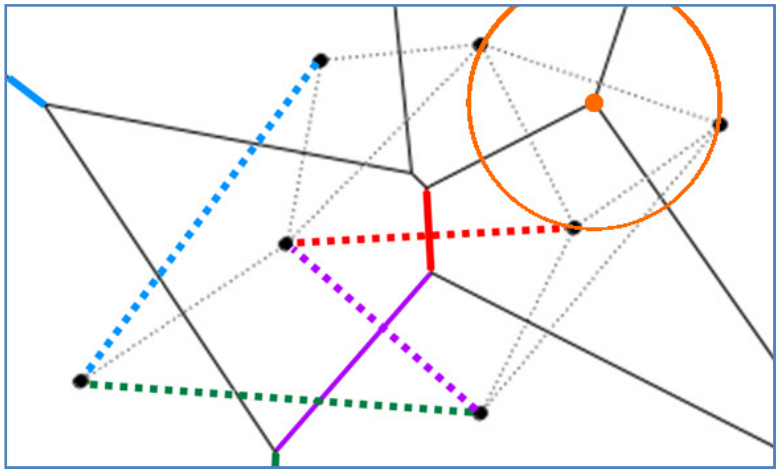


**Figure** 3.4Voronoi diagram (solid line) and Delaunay triangulation (dotted line) of a set of points. Edges with the same colour are each other's duals

Based on this property Ledoux (2006) proposed and implemented a robust algorithm that constructs exact 3D Voronoi diagrams, from the Delaunay tetrahedralization (3D generalization of the Delaunay triangulation). It also allows for points to be inserted, removed and moved within that 3D Voronoi diagram, without having to recreate the entire structure.

### 3.1.1. The Voronoi diagram and the Delaunay triangulation in 3D, and their duality

In this section and in the entire thesis in general, many concepts are illustrated using 2D visualizations. This is because of the fact that 2D images are easier to visualize on paper or a screen, but also because it is conceptually simpler to explain the different concepts and methods in 2D. In most cases these can be generalized into three dimensions, but in order to fully grasp all concepts, theories and methods, it is good to understand the following relations between the different 2D and 3D objects.

In 2D, Voronoi cells are separated from each other by boundaries that consist of line segments. The Voronoi cell consists of a convex polygon without holes. In 3D, these boundaries consist of planes, and the cells consist of convex polyhedral (also without holes). In Figure 3.5, an example of the 3D Voronoi diagram is shown. It is not easily interpreted, but this issue will be covered in Section 5.1.

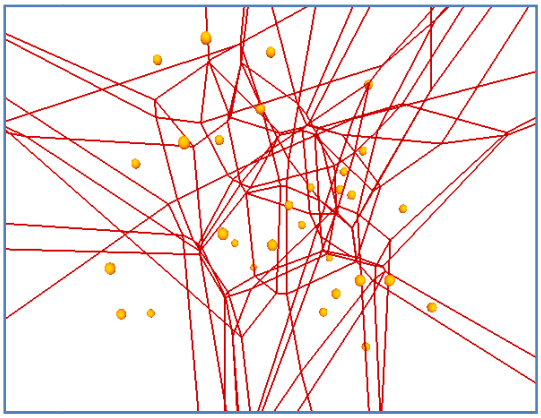


**Figure** 3.5 Example of the 3D Voronoi diagram. The yellow spheres are the seeds.

In the same way that in 2D the creation of a Voronoi diagram can be seen as the 'expanding circles' concept (Figure 3.1), in 3D this can be seen as spheres that expand from each seed, becoming flattened wherever the spheres touch each other. In Figure 3.6, a Voronoi polyhedron is shown. The red dot represents the seed.

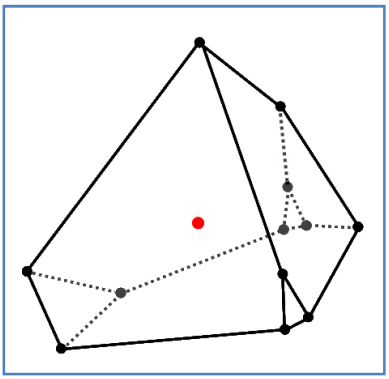


**Figure** 3.6 Example of a 3D Voronoi cell, or polyhedron. The red dot represents the seed.

It is important to notice that the lines connecting neighbouring seeds in a 2D Delaunay triangulation cannot be translated into planes in 3D, as is the case with the Voronoi diagram. In the Delaunay tetrahedralization, the seeds are also connected by edges. However, the triangles in a 2D Delaunay triangulation translate into tetrahedra for the Delaunay tetrahedralization. These tetrahedra consist out of 4 points, and can be thought of as 4-faced pyramids. A Delaunay tetrahedralization of a small set of points is shown if Figure 3.7, in which a tetrahedron is highlighted.

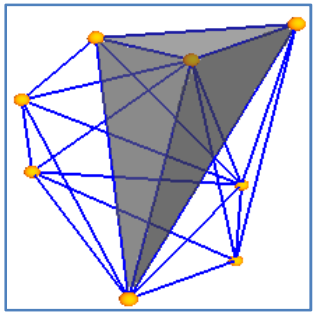


**Figure** 3.7 Example of a Delaunay tetrahedralization. The highlighted tetrahedron shows the 4 faces and 4 vertices of a Delaunay tetrahedron.

Duality in 3D

The dual relationship between the 3D Voronoi diagram and the Delaunay tetrahedralization is also present, as it is in the 2D representations. In this case however the dual relationships are as follows:

- A polyhedral in the Voronoi diagram is dual to the vertex in the tetrahedralization.
- A face in the Voronoi diagram is dual to an edge in the Delaunay triangulation
- A Voronoi edge is dual to a face of a Delaunay tetrahedron
- A Voronoi vertex is dual to a Delaunay tetrahedron.

As well as in 2D, the duality implies that if one knows the 3D Voronoi diagram, the Delaunay tetrahedralization can be derived, and vice versa. To understand this basic principal is imperative for understanding the difficulties in creating the exact 3D Voronoi diagram.

### 3.1.1. Why is the 3D Voronoi diagram difficult to implement?

In essence, the postulated algorithm will first compute the Delaunay tetrahedralization of a set of points, and from the Delaunay tetrahedralization, it will then infer the Voronoi diagram based on the dual relationship between the two structures. This is relatively efficient in most cases, but there are situations in which points are distributed in such a way that additional checks and calculations are required. These cases are also referred to as degenerate cases. An example of such a degenerate case is when four points lie on the same circle (cocircular) in 2D, or when 5 points are cosphrerical in 3D. This would yield a Delaunay triangulation that is not unique (there two different Delaunay triangulations possible), but the Voronoi diagram is still unique (Ledoux, 2006). Although the Voronoi diagram is unique, the fact that the Delaunay triangulation is not, has implications for further calculations when, for instance, removing points from the Voronoi diagram.

These degenerate cases make it very difficult to make an algorithm that is robust (and efficient), and also enables the user to dynamically add, move or remove points (Sugihara and Inagaki, 1995; Devillers and Teillaud, 2003; Ledoux, 2006). The fact that this is so difficult to achieve makes it worthwhile to see if the raster structure can increase efficiency or robustness.

## 3.2. VORONOI DIAGRAM IN RASTER SPACE

### 3.2.1. RASTER SPACE

The discrete Voronoi diagram is already explained slightly in the introduction as a representation of the Voronoi diagram in raster space. In this case, raster space is a space which is tessellated into regular cells, called pixels in 2D and voxels in 3D. The polygons (2D) and polyhedra (3D) of which the exact Voronoi diagram consists, are also present in the discrete 3D Voronoi diagram, only now these are represented by a group of pixels (2D) or voxels (3D) which share the same value.

Where vector space is essentially infinite (unless boundaries have been explicitly established), raster space has to be finite in order to work with it, otherwise pixels cannot be given indices, and no operations can be performed on the raster. By setting the boundaries for the raster, one automatically creates an image. This image then consists of a fixed amount of pixels between the boundaries that are situated in a certain order, and equals a 2D -or 3D- array. Because of this order of pixels within the image or array (spatial coherence), the efficiency of storing an raster image can be improved dramatically, since only the value of pixels have to be stored, instead of both location and pixels (de By, 2000). There are several spatial structures that decrease storage space of a raster such as quadtrees (2D) or octrees (3D) and various space filling curves (van Oosterom, 1999; Burrough and McDonnel, 2006), by using the spatial coherence of a raster image.

One key factor in the popularity of raster images in GIS and in computer software in general, is that it makes use of discrete packets of information. Information in a computer is always processed in a discrete way: on the lowest level of computer processing, only the values 0 or 1 are used. The discrete nature of the computer makes it an excellent environment to store and manipulate raster images. Also, when an image is generated on the computer monitor, this image is made up of pixels. These are the reasons that objects created in vector space very often have to be discretized before they can be manipulated or visualized.

Also, the fact that a raster image is in essence an array adds to the usefulness and popularity of raster images. The array is a native data structure in many computer programming languages, and many tools for manipulating, analyzing and storing arrays haven been developed, and are integrated within these languages. This makes it relatively easy to use arrays in developing tools.

#### Raster terminology

In the rest of this thesis, some expressions and terminology concerning the evaluation of pixels will be mentioned repeatedly. In order to avoid confusion, these expressions will be explained here. The process of giving a pixel the same ID value as the seed which is closest by, is called '*assigning a pixel an (ID) value*'. This act can also be referred to as a seed '*claiming a pixel*'. These expressions are used frequently during the rest of the report. Another situation that is often referred to in the thesis is the event in which a pixel is situated at exactly the same distances from two or more seeds. In this case a

pixel is said to be equidistant to two or more seeds. This situation is referred to as a '*tie*' or a '*conflict situation*'. If a pixel has been assigned an ID value that does not belong to the closest seed, but to another seed, this means that the Voronoi diagram is not correct, and the pixel is said to have an '*invalid (ID) value'*, or has been '*wrongfully'* or *'incorrectly claimed'.*

### 3.2.2. Different Metrics

As mentioned earlier, a Voronoi diagram is a tessellation of space. This tessellation is based on distance: all locations for which the distance to a seed $p_i$ is smaller than the distance to any other seed are associated with seed $p_i$.

In ordinary everyday life, distance is usually expressed using Euclidean distance. The distance $d$ between point *a* and *b* is the distance of a straight line between those points, which can be calculated based on the Pythagorean Theorem:

$$d(a,b) = \sqrt{(a_x + b_x)^2 + (a_y + b_y)^2} \qquad \text{Eq. 2}$$

(Worboys and Duckham, 2004). But distance can also be described in different ways. For instance, consider Figure 3.8.

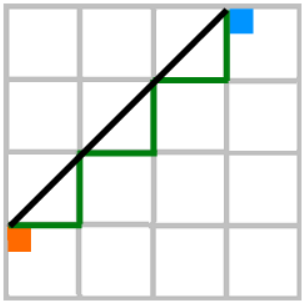
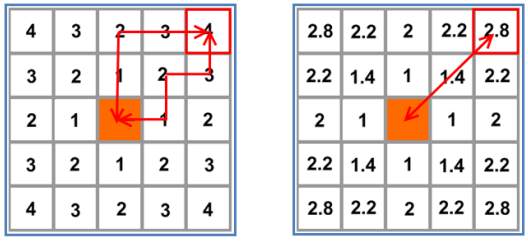


**Figure** 3.8 Euclidean distance (black line) vs. Manhattan distance (green line)

This illustration show 2 stores (orange and blue) in a city, where the white squares are city blocks (50 m. x 50 m.), and the grey lines represent roads. The black line illustrated the Euclidean distance which is approximately 212 m. However, if one had to drive by car, the distance would be 6 city blocks, which equals 300 meters.

This concept can easily be transferred to a raster image. Figure 3.9 a and b show a small 5 x 5 pixel grid. The numbers in each pixel represent the distance from that pixel to the orange pixel in the centre. In Figure 3.9a, the so-called Manhattan, or city block distance is used (named after the typical layout of Manhattan, New York (Worboys and Duckham, 2004). In Figure 3.9b, the Euclidean distance is used. In Figure 3.9a the distance between the orange pixel and the red lined pixel is 4, in Figure 3.9b, it is 2.8.




**(a)** **(b)**

**Figure** 3.9 Differences in distance between Manhattan **(a)** and Euclidean distance**(b)**.

From the above example, it is clear that distance can be defined in different ways. A space in which distance is defined according to the conditions given hereafter is called a metric space.

Suppose there is a set of points, called S. S can be called a metric space if for S there is a distance function d(s, t), where s and t are a pair of ordered points, that returns a distance that meets the following criteria (from (Worboys and Duckham, 2004):

1) **For every pair of ordered points $s, t$ in $S$:**

   $d(s,t) = 0$ when $s$ and $t$ are identical, and

   $d(s,t) > 0$ when $s$ and $t$ are distinct points.

2) **For every pair of ordered points $s, t$ in $S$:**

   $$d(s,t) = d(t,s)$$

   In other words, the distance from s to t must be the same as the distance from t to s.

3) **For every triple of ordered points $s, t, u$ in $S$:**

   $$d(s,t) + d(t,u) \geq d(s,u)$$

   In other words, the summed distance from $s$ to $t$ and from $t$ to $u$ is always equal or larger than the direct distance between $s$ and $u$.

There numerous methods available. The metrics most appropriate for this research are the Euclidean and Manhattan metrics. The distance function for Euclidean metric space is given in Equation 2. The distance function for 2D Manhattan metric space is:

$$d(a,b) = |a_x - b_x| + |a_y - b_y| \qquad \text{Eq. 3}$$

Manhattan metric space is also called $L_1$ metric space, whereas Euclidean metric space is called $L_2$ metric space. There is another metric called the $L_\infty$ metric space. The distance function for this metric is defined as follows:

$$d(a,b) = max[|a_x - b_x|, |a_y - b_y|] \qquad \text{Eq. 4}$$

In other words, it states that the distance between $a$ and $b$ is the maximum of the two absolute distances between the x coordinates and y coordinates of both points.

To illustrate the difference in metrics, one other example is shown. In Figure 3.10a a set of consecutive circles is drawn around a point pixel in Euclidean metrics in a raster. In Figure 3.10b the same consecutive circles are drawn, except now for Manhattan metrics. Figure 3.10c shows the circles in $L_\infty$metric.

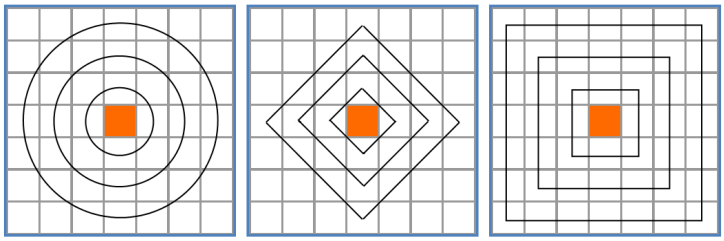



**Figure** 3.10 Set of consecutive circles represented in Euclidean **(a)**, Manhattan **(b)** and $L_\infty$metrics **(c)**.

The different shapes in Figures 3.10 a – c essentially represent distance isolines. Every point on such a line is located at the exact same distance from the seed as every other location on the line, measured in the corresponding metric. A circle in Euclidean distance becomes a diamond shaped figure in the Manhattan metric, and a square in the $L_\infty$metric. Going from Figure 3.10a to b, and to c can be seen as a distance conversion. The process of converting a raster from Euclidean distance into Manhattan distance is called distance transformation. How distance transformation and different metrics can be helpful will be made clear in Section 3.3.3.

It is important to realize that when modelling real-world situations in a raster, the conversion from real world object to pixel representation can introduce errors and issues concerning accuracy. For instance, a point might fall in the top-left corner of a pixel, but it is represented by the entire pixel. Also, the orientation of the grid can have an influence in the measured distances when using the Manhattan and $L_\infty$ distances.

## 3.3. Methods for creating the discrete Voronoi diagram

Creating a discrete Voronoi diagram can be done through several mechanisms. Among these are:

- Nearest neighbour sweep circle method (Schueller, 2007)
- Dynamic distance transformation and dilation (Li, Chen et al., 1999)
- GPU based methods (Hsieh and Tai, 2005; Rong and Tan, 2006; Majdandzic, Trefftz et al., 2008)
- Using tree-structures (Park, Linsen et al., 2006; Zhao, Li et al., 2008)
- The incremental method (Fuchida, Kashima et al., 2005)

These methods will be presented in more detail, followed by a discussion concerning the general advantages, possibilities and applications. Most of these methods are explained in 2D because they have been developed in 2D and because this makes it more easily visualized. However, they can be generalized into three dimensions.

### Implicit and explicit methods

The above mentioned methods can roughly be divided into two groups which I call the implicit method (such as those using dilation, Section 3.3.4), and the explicit method (like the naïve method, Section 3.3.2).

Using the explicit methods, each pixel in the Voronoi diagram is evaluated, and the distance to the closest seed is determined through ordinary distance calculation methods. The risk that a pixel is assigned an incorrect value is very low. Even if one pixel is evaluated incorrectly, values of other pixels do not depend on the erroneous pixel, so the error does not progress further. This method is not very efficient in the case of the naive method (Section 3.3.2). However, the efficiency or speed of the algorithm might be increased by using auxiliary data structures, but these structures have other disadvantages, such the need as additional storage space and keeping the structure up to date. One example of an explicit method with an auxiliary data structure is shown in Section 3.3.7.

On the other hand, implicit methods will generally assign a value to a pixel based on the value of their neighbours. In other words, based on pixel adjacency, it is implicitly assumed that a certain ID value can be assigned to a certain pixel. Only in situations in which it is not entirely certain that the correct value is assigned to a seed, more checks need to be performed in order to make sure the Voronoi diagram is correct. These situations will occur near the boundaries of the Voronoi cells. Although this seems to be more efficient than explicit methods, it is imperative to implement an algorithm that decides correctly whether or not a pixel can implicitly be assigned an ID value, or if more checks are need. If this is not the case, and a pixel is assigned a wrong ID value, the erroneous value might be distributed throughout the Voronoi diagram.

In Figure 3.11, the difference between the implicit and explicit methods is illustrated.

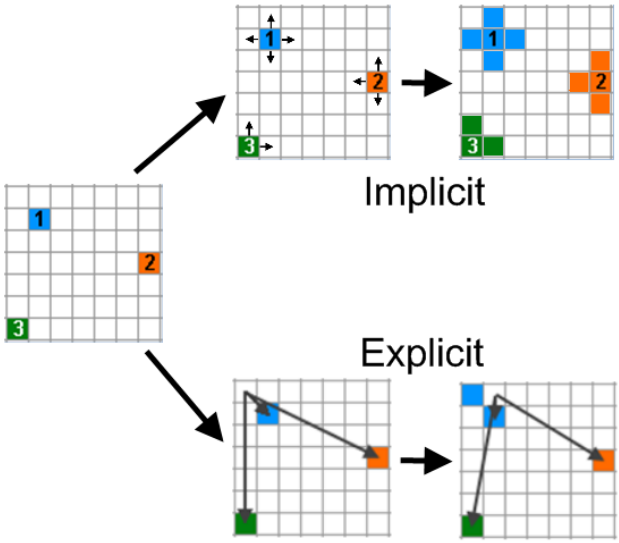



**Figure** 3.11 The difference between the implicit and explicit methods: The implicit methods implicitly assumes that neighbouring pixels have the same value. The explicit methods explicitly calculate the value of each pixel.

### 3.3.1. From exact to discrete Voronoi diagram

The first method for which all components are present is to create an exact Voronoi diagram, and convert it to a raster structure. This can be done by polling the exact Voronoi diagram at the locations of the pixels/voxels in the discrete Voronoi diagram, and thus creating a raster image that analogue to the exact Voronoi diagram. Obviously, this is not the desired method, since it requires the user to implement the exact Voronoi diagram first. Since this thesis is concerned with not having to use the exact Voronoi diagram, this method is not usable. However, it is mentioned here to give an overview that is as complete as possible.

### 3.3.2. The Naïve Method

The first method for constructing a discrete Voronoi diagram directly, is probably one of the most intuitive but also one of the least efficient methods available. It belongs to the explicit methods, and evolves from the general notion that every pixel/voxel in a Voronoi cell is assigned the value of the closest seed. This is easily done by checking which seed is closest for every pixel in the image. Although this is a very intuitive method, it is quite inefficient. In Figure 3.12 the naive method to create a discrete Voronoi diagram is illustrated.

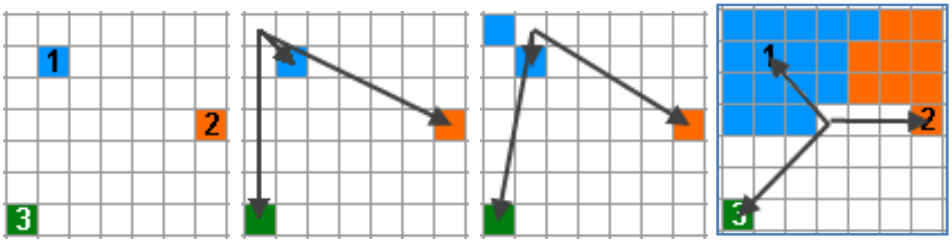



**Figure** 3.12 **(a)** A 7x7 pixel raster with three seeds (ID's 1,2 and 3). **(b)** The distances from the first pixel to all seeds are determined. **(c)** The closest seed has '*claimed*' the first evaluated pixel. The second pixel is evaluated. **(d)** This process is done for all empty pixels, from left to right, top to bottom.

It shows an image of 7x7 pixels, in which 3 seeds are placed, each with a unique ID value (1,2,3). To make it more clear, each seed is also represented by a colour. In order to create the discrete Voronoi diagram, for each empty pixel in the raster, the closest seed must be determined. This is done by, for each pixel, calculating the distance from that pixel to all seeds in the image, and assigning the ID value belonging to the closest seed to that pixel.

For large images and/or large number of seeds, it will be very slow to calculate the Voronoi diagram, especially in 3D. Another major disadvantage is when trying to add, move or remove seeds; this method requires the entire Voronoi diagram to be rebuilt after adding, moving or removing a seed.

To make the algorithm more efficient, a spatial indexing techniques such as the R-tree or the kd- tree can be used (Rigaux, Scholl et al., 2002). However, the main disadvantage of using a spatial index is that an auxiliary data structure has to be built first, which also has to be kept up to date. This requires relatively much time and memory space.

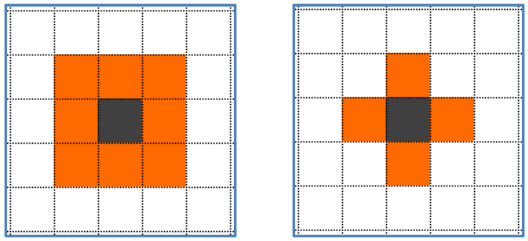
### 3.3.3. Nearest Neighbour Sweep Circle Algorithm

In Schueller (2007), a method is proposed for creating a 2D discrete Voronoi diagram that is similarly intuitive to the naïve method, but much more efficient. It is based on the principle of expanding circles, explained earlier in the section concerning the exact Voronoi diagram (Section 3.1). This principle can be of use in making an algorithm more efficient because the fact that circles are used implicitly means that only the pixels at the boundaries between two or more Voronoi cells have to be checked to which seed they belong. In other words, as long as circles of two seeds do not touch, all pixels within that circle are correctly assigned to the corresponding seed.

The algorithm starts by setting all pixels to 0 (marking them as not claimed), and then setting pixels which represent the seeds to their corresponding ID-values. The initial radius of the circle $r$ is set to a positive value, as is the radial increment $\Delta r$. The radius index (which keeps track of how often the radius has expanded) is set to 1. The algorithm then enters a loop which can be expressed as follows:

```
1    FOR each claimed pixel P:
2        x = seed with value of P
3        Claim all unclaimed neighbours that are within distance r + ℓ · Δr of seed x
4        If a neighbor is already claimed by a seed y THEN:
5           IF P is closest to x THEN:
6              assign neighbor to x
7           ELSEIF P is equidistant to x and y THEN:
8              assign neighbor to seed with smallest index
9    NEXT
```

To assess which pixels are neighbours of a claimed pixel ***P***, Schueller proposed 3 neighbourhood sets, among which are the Moore neighbourhood and the Neumann neighbourhood. In Figure 3.13a and b, the Moore and Neumann neighbourhoods are shown. These are also known as 8-neighbourhood and 4-neighbourhood respectively. The grey pixel (central pixel) in the middle represents the claimed pixel ***P***, the orange pixels represent those that are its neighbourhood pixels.



**(a)** **(b)**

**Figure** 3.13 Two different neighbourhoods. The Moore neighbourhood **(a)**, and the Neumann neighbourhood **(b)**

When considering these two neighbourhoods in 3D, the Moore neighbourhood consists of a 3x3 voxel cube in which the central voxel (2;2;2) is empty. When dilating a pixel using the 3D-Moore neighbourhood, the resulting image will be a cube. The 3D Neumann neighbourhood consists of a 6-voxel object, where all 6 faces of the central voxel are adjacent to 1 of the six voxels. Dilating the 3D Neumann neighbourhood results in a diamond shaped object. There are also other neighbourhood sets available such as the 16-connected and 32-connected described in van Bemmelen, Quak et al. (1993). These will be discussed in Section

Discussion

Schueller notes that the algorithm can be made more efficient by not considering points whose neighbours have all been claimed already. But even when taking that into consideration, one major shortcoming in terms of efficiency for this algorithm can be identified. It is located in the statement in line 3 in the pseudo-code. To check whether a neighbouring pixel falls within the boundaries of $r + \ell \cdot \Delta r$ of seed *x* is costly, since this has to be done for every pixel that is claimed, at least once. The check has to be performed at least twice for pixels that lie on the boundary between Voronoi cells. This

means that the advantage that comes with using the sweeping circle or expanding circle principle is not exploited. The sweep circle principle would be an excellent option when considering that using Figure 3.13a as a neighbour set could yield a circle in the $L_\infty$metric. The same goes for Figure 3.13b, but then for the Manhattan metric. Using each of the neighbourhood sets to create a Voronoi diagram in the corresponding metric would be advantageous, since no checks would have to be made to see if a pixel falls within the radius of the next circle, because the claimed pixels would actually define the circle. However, since the Schueller is looking for a Euclidean Voronoi diagram, the advantage of the automatic distance transformation gained by using the either of the neighbourhood sets is partly lost. More on implicit Voronoi diagram creation algorithms is presented in Section 4.2.1.

Also, in the event of a tie, that is when a pixel is equidistant from two or more seeds, the seed with the smallest index number is assigned to the pixel. This will give a systematic error, and Schueller gives no explanation as to why this rule of assigning is chosen. Other possibilities, such as arbitrary assignment of the boundary pixels, will be discussed in Section 5.3.2.

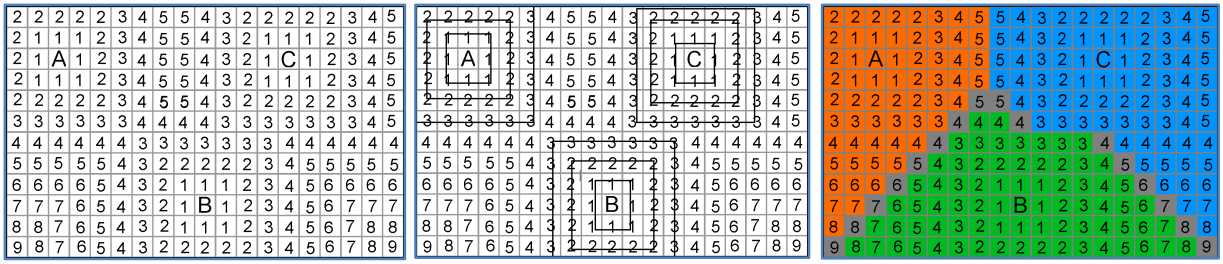
When looking at the different neighbourhoods used in the article, two properties are distinguished that are of importance: accuracy and efficiency. On both counts, using the Neumann neighbourhood yielded better results. Especially in the vicinity of corners of the exact Voronoi diagram the Neumann neighbourhood shows fewer errors in assigning pixels.

Adapting the algorithm so that seeds can be added will be relatively easy with this algorithm. The only thing that needs to be done is to insert the new seed as a pixel, and dilate this seed. The algorithm will automatically stop once the new Voronoi cell is added. Also, moving and removing a point will not be difficult; by marking all the pixels of the Voronoi cell to be deleted, the surrounding seeds can be expanded, thereby overwriting the marked pixels.

All in all, this algorithm does look promising, and will be taken into account when designing my own algorithm.

### 3.3.4. Dynamic Distance Transformation and dilation

Another 2D discrete Voronoi diagram algorithm is presented by (Li, Chen et al., 1999) Li, Chen et al. (1999). In this article, the authors begin by stating that Euclidean distance is not preferable, because distances between two pixels are in decimal form, which is "*inconvenient to use in raster mode and a distance in integer number is more desirable and thus normally employed."* They therefore propose a method based on the concept of distance transformation Section 3.2.2. This concept can be used to create an approximation of the exact Voronoi diagram in raster space. An example of this is seen Figure 3.14.



**(a)** **(b)** **(c)**

**Figure** 3.14 The creation of a discrete Voronoi diagram using distance transformation. In **(a)** The $L_\infty$ metric is used to assign distances to the closest seed. In **(b)** consecutive circles are drawn until these collide. In **(c)** the resulting Voronoi diagram is shown, with each seed represented by a colour (orange, blue and green). The grey pixels are pixels represent pixels that are equidistant from one or more seeds.

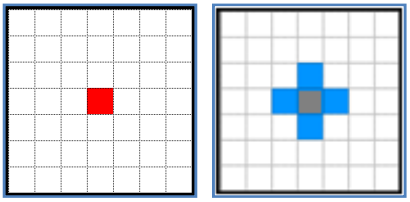
In Figure3.14a, three seeds are inserted, namely seeds A, B and C. Using the $L_\infty$ metric (chessboard distance), the distance from these seeds to all other pixels is evaluated. This leads to a raster with distance values. In this raster, contour lines can be drawn (Figure 3.14b). By joining the pixels containing the highest numbers for each seed, one attains the Voronoi diagram for the specific metric (Figure 3.14c), in this case the $L_\infty$ metric. In this Voronoi diagram however, the grey pixels represent the pixels that are equidistant to two or more seeds. Also in this article there is no mention of how such pixel should be treated. In the chapter concerning the algorithm I created (Chapter 4) this issue will be discussed in more detail. The authors state that the concept of connecting the pixels with the highest distance for each seed can be replaced by a different mechanism, namely the morphological dilation operator.

The dilation operator is an operator that is frequently used in the field of computational geometry, digital image processing and analyzing, medical image analysis and many other fields (Wu, Xu et al., 2005; Jamil, Sembok et al., 2008; Tang and Chen, 2008; Martini and Wu, 2009). The mathematical formula for the dilation operation is (Haralick, Sternberg et al., 1987):

$$A \oplus B = \cup_{b_\varepsilon B} A_b \qquad \text{Eq. 5}$$

where A is the original image, which is dilated using B, which is the structuring element.

It is the opposite of morphological erosion, operations which are both used on binary images. Binary images are images in which pixels can have 2 values: 1 or 0. Both the dilation and erosion operations are based upon two elements: the main image (Figure 3.15a) and the structuring element (Figure 3.15b). The structuring element consists of a 'central cell' (grey) and 'stamping' cells.

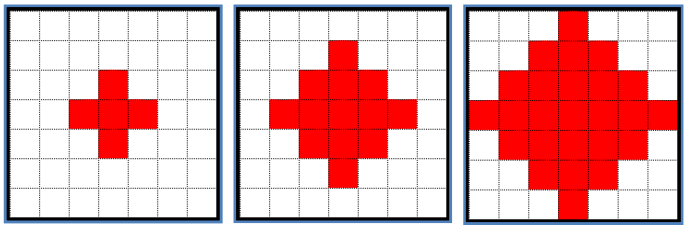


(a) (b)

**Figure** 3.15 **(a)** The image to be dilated, **(b)** the structuring element

In the example in Figure 3.15a, the image is a 7x7 raster, in which all cells are 0 except the centre cell, which is marked 1 (red). In the structuring element in Figure 3.15b, the blue pixels represent the pixels with value 1. When the dilation operation is performed, the central cell is placed successively over all pixels marked 1 in the main image. Each time the structuring element is placed over a pixel in the image, the pixels in the image that overlap with the blue pixels of the structuring element are marked 1. This operation can be performed an infinite amount of times, each time expanding or dilating

Figure 3.16a shows the image after it has been dilated once, Figure 3.16b and c show the main image after having been dilated two and three times respectively.



(a) (b) (c)

**Figure** 3.16 The image after one **(a)**, two **(b)** and three rounds of dilation **(c)**.

In the study by (Li, Chen et al., 1999) Li, Chen et al. the authors want to use the concept of dilation to find the distance contour lines (as in Figure 3.14b) in order to determine the Voronoi cell. Also they want to approximate Euclidean distance as well as possible. In order to find these contours, they make use of the concept of the expanding circles, which has a great advantage that until the circles touch (or in this case contour lines), all pixels within the circle or contour line are implicitly assigned to the corresponding seed. This means that only when the contour lines of two or more seeds touch, does a check need to be performed as to which seed they belong. This can increase performance drastically.

In order to find the correct final contour lines (Voronoi boundaries) that approximate the Euclidean distance as good as possible, the authors state that the expanding contour lines should resemble a circle as much as possible. In order to do this, they combine the concept of distance transformation and dilation by creating several different structuring elements. These different elements are then placed at specific locations in such a way that the new image resembles a Euclidean circle as much as possible. This technique is then adapted so that it can dilate other features beside points (such as lines and areas)

as well, in such a way that the final image represents a Voronoi diagram of those features. The creation of Voronoi diagrams for other objects than points is indeed an interesting topic, and will be discussed further in Section 5.2.

Discussion

The authors state that using Euclidean distance in raster environments is not preferable. Therefore they propose a method to create a Voronoi diagram in raster distance. But the algorithm proposed makes use of Euclidean distance to calculate whether or not the dilated image represents a Euclidean circle as close as possible. This contradicts the goal of the authors, and is not very efficient. Also I see no objection to using Euclidean distance, especially when the outcome needs to resemble the Voronoi diagram in Euclidean distance as good as possible.

Another remark can be made about the criteria that the dilated image must resemble a Euclidean circle as much as possible. Although this does give the advantage of knowing explicitly which points are closest to the seed corresponding to the circle, as soon as the circles (or contour lines) start to meet each other, checks have to be made as to which seed is closest.

A different possibility would be to use one structuring element instead of several to approximate a circle. Although this will require more of these checks, since the dilating object does not approximate a Euclidean circle as well. But if only one structuring element would be used, the calculations that are necessary to evaluate the differences between the dilated circle and the Euclidean circle would also not be necessary. A further discussion on the shape of the structuring element will be given in Section 5.3.

For creating correct Voronoi diagrams of other object (lines, areas), it seems to be an effective algorithm, since it is able to effectively dilate objects of different shape, without having to calculate the closest object for each pixel. However, to use this method for the creation of a Euclidean Voronoi diagram of point features seems not to be very efficient. This is because the dilated shapes need to be checked in every round to see what structuring elements are needed to get a shape that is as close to a circle as possible. Also, this algorithm shows possibilities in adding, moving and removing points, similar to the nearest neighbour sweep circle algorithm. Based on these advantages, this algorithm will also be taken into account for the creation of a discrete 3D Voronoi diagram creation algorithm.

### 3.3.5. GPU BASED METHODS

In a personal computer, computer code is normally executed by the CPU (Central Processing Unit). It receives instructions (calculations), processes them, and returns information. However, since several years, the GPU (Graphical Processing Unit) is a standard feature in most PCs. It is an auxiliary processing unit, next to the CPU, and it is specially designed to render graphics by executing all the necessary calculations (image transformations, shading, and texture application). It can do this very effectively, since it is specifically designed to perform operations such as linear algebraic operations. These calculations are very complex, and using a GPU drastically cuts the load on CPU, ultimately making the

computer perform many times faster. The later versions of the GPU are also designed to perform parallel calculations, which means that they can make several computations at the same time (Rong and Tan, 2006). This is another factor in the major speed increase that the GPU provides.

As already mentioned in the research objectives, using the GPU is not within the scope of this research. In order to utilize the GPU, my knowledge of, and skills in programming would have to be stretched far beyond what they are now in order to perform the low-level programming necessary for the usage of the GPU. This would be far too time-consuming for this thesis. This does not mean, however, that the algorithms and ideas in the presented papers cannot help in creating a new algorithm. Therefore, articles that might be relevant for this research project are discussed below.

Several people have devised ways to create a discrete Voronoi diagram with the aid of a GPU (Hsieh and Tai, 2005; Rong and Tan, 2006; Rong and Tan, 2007; Majdandzic, Trefftz et al., 2008). A small, relevant selection of these will be summarized.

It is important to note that most GPU based methods are not applicable directly for the creation of the 3D discrete Voronoi diagram. However, Rong and Tan (2007) present a method to create a 3D Voronoi diagram using the GPU.

### A simple GPU-based approach

In Hsieh and Tai (2005) (Hsieh and Tai, 2005), the authors rely mainly on the power of the GPU. They use a variation of the 'naïve method', in which the distance between a pixel and a seed is calculated for each seed, instead of each pixel. Each pixel keeps track of the current nearest seed and the distance to that seed. If a seed is closer that current nearest seed, the pixel is updated. The implementation is slightly different from that of the naïve method, but the efficiency is still similarly poor. The only thing that increases the speed of the algorithm is the sheer power of the GPU.

### Jump Flooding Algorithm (JFA)

In Rong and Tan (2006, 2007) (Rong and Tan, 2006; Rong and Tan, 2007), the authors propose a method called the Jump Flooding Algorithm (JFA) in which they utilize the parallel programming options of the GPU, in combination with a efficient algorithm. The concept of jump flooding can be described as follows:

In a raster of $n$ x $n$ pixels, the pixels that represent the seeds are assigned a unique ID value. In the first round after this initial state is set, each seed at $(x_i, y_i)$ will pass on its ID to the pixels at $(x_i + m, y_i + n)$, where $m, n \in [[-k, 0, k]]$, and $k = n / 2$. This is done for each seed simultaneously. This means that depending on the location of the seeds, a pixel might be assigned ID's of several different seeds. Of these seeds, the one that is closest to that pixel is chosen, and recorded for that pixel. In each subsequent round, this process is repeated, but each round the step length $k$ is divided by 2, in other words, the step length $k$ will be $n / 2$, $n / 4$, … $n/n = 1$, for each subsequent round.

To illustrate this process, the Figures 3.17a-d show the concept of jump flooding for one seed in an 8 x 8 raster.

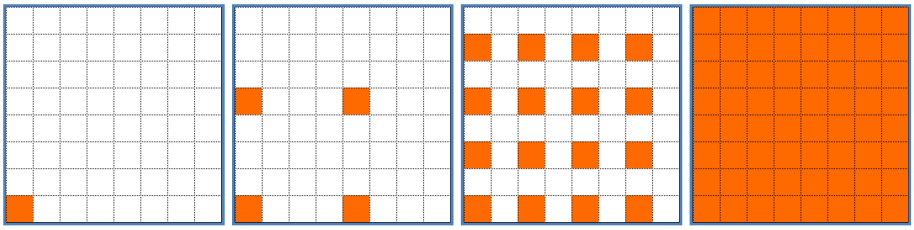



**Figure** 3.17 **(a)** Start of the image, n = 8, k = n/1, **(b)** k = n/2, **(c)** k = n/4, **(d)** k = n/8. Recreated from Rong and Tan (2006).

One disadvantage of this method is however that it is somewhat prone to errors. Because the algorithm depends on the propagation of values from pixels to other pixels, an erroneous image may be formed when a pixel receives the incorrect value. This is referred to as a pixel or site being 'killed', and it would cease the propagation of the correct value to other pixels. This is partly solved in the revised algorithm 1+JFA in Rong en Tan (2007), making this algorithm more accurate.

In that same paper, another implementation of the JFA and 1+JFA is presented. This variation allows for the creation of 3D discrete Voronoi diagrams, without the use of the GPU. The GPU is not used in this implementation, because the functionality of the GPU did not allow for this method to work in 3D. Thus, the CPU is the only available option. This variation in 3D works on in slice-by-slice manner, in which the 3D raster is sliced into 1 voxel-thick slices either in the x-, y- or z-plane. These slices are then handled separately. This 3D version of the (1+)JFA relies on the property that the seed does not need to be represented in the exact correct location in the raster in order to gain a correct Voronoi diagram (property 1 in Rong and Tang, 2007), as long as no pixels are 'killed'. Simply put, this is because in each subsequent round, every pixel that is assigned a certain ID value is checked so that only the closest seed is assigned to that pixel. If a seed is placed at an offset, distance checks at each pixel should prevent pixels in the next round to get a wrong value, unless pixels get killed. Therefore the authors propose to orthogonally project the seeds on each slice, and run the 1+JFA for each slice. When calculating and checking distances, 3D Euclidean distances are used, ensuring that the correct seeds are assigned to the respective voxels. Afterwards, the slices can be added together again, recreating the entire 3D Voronoi diagram.

The authors claim that the general concept of the JFA is applicable not only to points, but also to other objects, such as lines, splines, etc.

Discussion

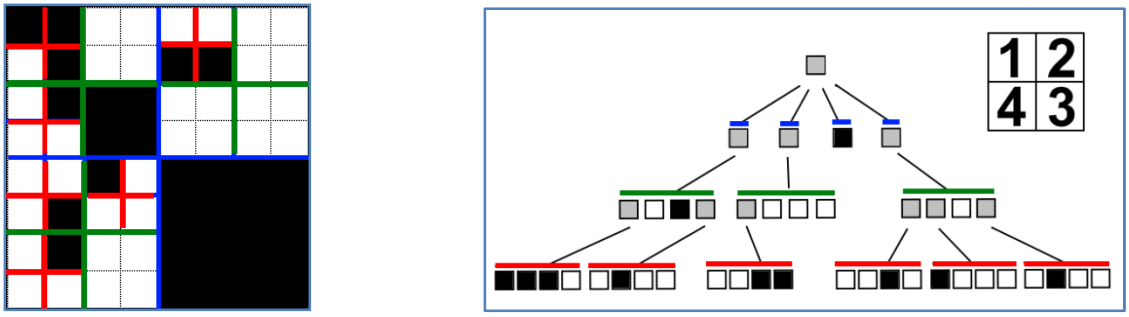
The execution of the JFA and 1+JFA is done within the GPU, which is obviously one of the main advantages of this algorithm. If this algorithm would be implemented using a serial processor such as

the CPU, it would not be an efficient algorithm at all. This is due to the fact that each pixel can receive a minimum of one, and a maximum of eight different ID values from pixels at ($x_i + m$, $y_i + n$), where $m, n \in [[-k, 0, k]]$. This means that the distance between a pixel and a seed is evaluated at least once, and at most eight times per pixel. By executing these calculations parallel (simultaneously) the increase in speed can be dramatic. But since all GPU related algorithms fall beyond the scope of this project, it means that this algorithm is not usable for this research project.

The 3D implementation, the slice-by-slice algorithm, however is suitable only for use with the CPU. This makes it much more interesting for this research project. But the main issue with this method is that the drastic increase in efficiency that the GPU yields is lost completely. The GPU can at present not be implemented to create the discrete 3D Voronoi diagram. Therefore, this method will not be taken into account when creating a discrete 3D Voronoi diagram algorithm.

### 3.3.6. A Hierarchical Raster Method for Computing Voronoi Diagrams based on Quadtrees

In (Zhao, Li et al., 2008) Zhao, Li et al., 2008, the authors present an algorithm that uses quadtree structures to ultimately create a 2D discrete Voronoi diagram. The quadtree structure is a way of representing and storing images, based on the recursive subdivision of the image into quadrants in such a way that the value of every pixel is recorded, but not every pixel is stored separately (Samet, 1982). In other words, it groups pixels together that have the same value and are situated in a square of any size (the smallest of which is a 1x1 square). Figure 3.18a shows an 8 x 8 raster image, in which the black pixels are 1 (filled), and the white pixels are 0 (not filled). The blue lines represent the subdivisions of the first level. The green lines represent the second subdivision. The red lines represent the third subdivision. Figure 3.18b shows the quadtree of this image. The square with the numbers 1,2,3 and 4 in Figure 3.18b show the numbering of the subsequent areas in which a square is subdivided. In the quadtree structure, white nodes represent empty regions, black nodes represent fully filled regions and grey nodes represent semi-filled regions. The first node in 3.18 represents the total image. Since not all pixels in it are black (semi-filled), it is coloured gray.




(a) (b)

**Figure** 3.18 **(a)** An 8x8 pixel image, showing the three levels of subdivision. Blue lines represent the first level, green the second and red the third level **(b)**Visualization of the quadtree structure, corresponding to the subdivision made in **(a)**. The square containing the numbers 1/4 show the sequence in which a square is subdivided.

By using a quadtree, images can be stored relatively efficiently, because groups of pixels can be stored as one entity, instead of having to store the value of all pixels separately. In the case of the image in Figure 3.18a, only 11 counts of black regions have to be stored, instead of 29 separate black pixels. The tree-structure itself also needs to be stored, but for even for relatively small raster, this will be an improvement.

This concept is adapted by Zhao, Li et al., and used in combination with the morphological dilation operator. They reason that when dilating seeds into Voronoi cells, it is only necessary to calculate distances between a pixel and two or more seeds for pixels near the boundaries of Voronoi cells. Pixels that are closer to the seeds might be assigned to a seed more effectively. This is done by creating a quadtree of the image, so that whole areas of pixels that belong to a certain seed can be assigned that value in one operation. These areas are represented by the black nodes in Figure 3.18b. When nearing the boundaries between two or more Voronoi cells, the subdivision is extended so that separate pixels can be evaluated.

### Discussion

One of the drawbacks of employing a structure like the quadtree is the large overhead needed for the creation of the quadtree. While storage space of a raster can be decreased dramatically by using a quadtree/octree structure, the overhead of memory space needed for the creation and manipulation of a quadtree (creating a Voronoi diagram for instance) is relatively high. Although this is the case, this does not mean that using quadtree structure cannot be an effective aid in creating a discrete Voronoi diagram. If one would try to create a discrete 3D Voronoi diagram, an octree could be utilized, which is the 3D version of a quadtree. In an octree, a cube is recursively divided into eight smaller cubes.

The concept which Zhao, Li et al. present does seem rather efficient, especially when having images in which the resolution is relatively high, or when the number of seeds is relatively low. In these situations,

it would be faster to be able to assign larger areas at once to a certain seed, than doing it one pixel at a time. Figure 3.19 zooms in on such a situation, where a normal dilation function is being executed. The dark colours represents the actual dilated Voronoi areas so far (using a Moore neighbourhood), where as the lighter colours represent the Voronoi cells as they should be at the end. In this case, calculation time could be decreased by assigning larger areas at once to the seeds. The black lines show the areas that each could be assigned to the blue seed in one round.

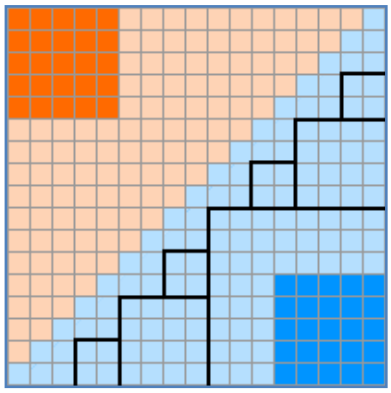


**Figure** 3.19. Enlarged portion of an image in which dilation is being executed (using a Moore neighbourhood). The black squares show the areas that could be assigned to the blue seed in one round.

The authors propose to do this using a quadtree structure. However, from the article it does not become clear how the quadtree is constructed when the pixel values are not known yet. In other words, how does the algorithm know where the boundaries between Voronoi cells are situated? Without this knowledge, a quadtree cannot be constructed in one round. While the concept is sounds attractive, it will not be utilized for this research project because of the limited explanations in the article, and because of the practical implementation issues concerning the octree in Python.

### 3.3.7. USING A KD-TREE

As already mentioned in the Section 3.3.1, using a spatial structure such as the R-tree or kd-tree can increase the efficiency of the naïve method greatly. The quadtree has also been mentioned, and in Park, Linsen et al. (2006) the kd-tree is used to create both 2D and 3D discrete 3D Voronoi diagrams. This is a recursive partitioning of space that organizes points in a k- dimensional space, by k number of keys. This is done as follows: A set of points is divided recursively at that point (node) which is the median of a certain property of those points (position in one dimension, spread, or any other), called the key value. All points with a key value lower than that of the node are assigned to the left sub-tree and all values large than the key value into the right sub tree. These sub trees are then divided in a similar fashion. The key in which the space is divided is chosen cyclically. This means the root subdivision is done by the $k_1$ key, the second subdivision by the $k_2$ key, up to the $k_{total}$ key, after which the k1 is chosen again (Bentley and Friedman, 1979; Yianilos, 1993). In this case, for a kd-tree in 2D, the first subdivision is done in the x-

dimension, the second in the y-dimension, the third in the x-dimension, etc. A graphical example of a 2D kd-subdivision is seen in Figure 3.20. For a kd-tree in three dimension, space would be alternately subdivided on the x, y and z dimensions.

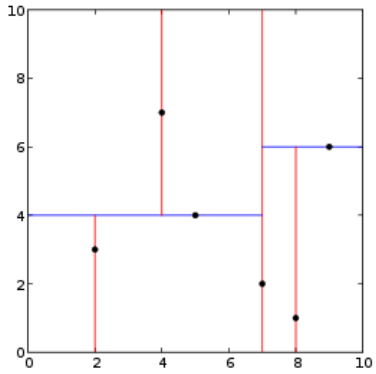



**Figure** 3.20 The decomposition of a kd-tree based on the points represented by the black dots. (Wikipedia, 2006)

Using the kd-tree to find the nearest neighbour of a point in a set of points can be very efficient (Bentley and Friedman, 1979; Yianilos, 1993; Talbert and Fisher, 2000; Park, Linsen et al., 2006; Liu, Chen et al., 2009). The kd tree can be implemented relatively easily. Using the data structure to find nearest neighbours of a point can be more difficult, but nonetheless possible.

Since the discrete Voronoi diagram is essentially a diagram that shows which seed is the closest neighbour for each pixel, creating the Voronoi diagram from a kd-tree of points is relatively straight forward. First of all, the kd-tree of the set of points must be created. Then for each pixel, a new location has to be inserted into the kd-tree after which the kd-tree can be queried for the location of nearest neighbour of that pixel. The ID value of this seed can then be assigned to the pixel. After all pixels have been evaluated, the Voronoi diagram is formed.

Normally, the creation of a kd-tree requires all seeds to be known from the start in order to get a balanced kd-tree. A balanced kd-tree is a tree in which each leaf is located at approximately the same distance (number of divisions) from the root of the tree (van Oosterom, 1999; Langetepe and Zachmann, 2006).

### Discussion

This very straightforward method is relatively efficient, but the overhead of this method is quite large, similar to the quadtree method. Another advantage of this method is that it explicitly calculates which seed is closest for each pixel, in other words, this is an explicit method, and if the nearest neighbour calculation is implemented correctly in this structure, there is very little chance of errors.

The kd- tree is not only a suitable data structure to use when searching for the nearest neighbour, but it also handles image scaling very well (Yianilos, 1993; Park, Linsen et al., 2006). This means that the set of points that is used to create the kd-tree can be scaled to the distance range [0, 1]. Any metrics range, including that of Euclidean distance can be scaled to the same [0, 1] range, so that after the kd-tree is created, the coordinates of pixels that need to be inserted into the kd-tree can be scaled identically, retaining the nearest neighbour relations. When creating the Voronoi diagram the results can be easily scaled into any other image size, without increasing the risk of errors.

One major disadvantage lies in the possibility of adapting this algorithm so that seeds can be added, moved or removed. The problem with this is not that inserting, moving or removing points in or from the kd-tree is highly time consuming, but to be able to perform a nearest neighbour search requires a balanced tree. After points have been inserted or removed, the tree becomes unbalanced, and has to be re-balanced. But the tree structure and the discrete Voronoi diagram are not dynamically linked; in other words, when updating the tree, the Voronoi diagram is not updated automatically. To solve this, at first glance, the entire Voronoi diagram would have to be recreated from the kd-tree, if the kd-tree is changed. A more innovative solution would be to somehow calculate what area of the Voronoi diagram is affected by the change in the kd-tree, and recreating only that area. However, calculating this area might not be easy, nor will it very efficient if many seeds were to be added or manipulated.

Based on the advantages and disadvantages, this method will also be looked at when creating a discrete 3D Voronoi diagram algorithm. There are several other tree-like data structures; however, the described quadtree and kd-tree are the most suitable for the required purpose. This is true for the kd-tree especially, since it is well equipped for nearest neighbour searching. Other available structures might work, but offer more functionality than is necessary. For an overview of more spatial index structures, see (van Oosterom, 1999)

### 3.3.8. THE INCREMENTAL METHOD

Fuchida, Kashima et al. (2005) presented an article titled “Constructing three-dimensional discrete Voronoi diagrams by the incremental method and application to self-organizing maps”. The paper is rather unclear in the description of the methodology, but one thing is certain: although the name of this method might be suggestive, this method has nothing to do with the incremental method used to create a Voronoi diagram in vector space (Guibas and Stolfi, 1983). I believe that the concept of this method is as follows in 2D: by stacking lines of a variable length and of a certain value together, areas (Voronoi cells) can be created. But the authors do not clarify in which way the length or value of these lines (or planes in 3D) can be created. They refer to a previous article, which is only available in Japanese.

Although the concept looks promising, there are too many unknown mechanisms to adapt this method and use it for the discrete 3D Voronoi diagram algorithm.

## 3.4. SUMMARY

In the beginning of this chapter, the exact and discrete 3D Voronoi diagram have been discussed in depth, and the duality between the Voronoi diagram and the Delaunay triangulation has been explained. Also, the concept of metric space is discussed, including the Euclidean, Manhattan and $L_\infty$ metrics.

Several existing methods for the creation of the discrete 3D Voronoi diagram have been discussed. Two different types of methods can be distinguished. Implicit methods assign values to voxels by assuming they carry the same value other voxels (usually the voxel next to them). This assumption makes it imperative that a good error-control mechanism is in place. Explicit methods explicitly calculate the value for each voxel. For this method, errors are less likely to occur, however, a good mechanism for these calculations must be in place in order to make the algorithm efficient.

The following methods for creating the discrete 3D Voronoi diagram have been discussed:

- The Naïve method:
  This is a very inefficient explicit method, which can be used in 2D and 3D. It is used purely as a test case for starting the discrete 3D Voronoi diagram algorithm development.
- *Nearest Neighbour Sweep Circle algorithm:*
  This implicit method is used in 2D, and is based on the principal of dilation. It aims to approach an expanding Euclidean circle as close as possible through dilation, however, the checks that are performed in order to do so are expensive.
- *Dynamic Distance Transformation:*
  This explicit method is also applied in 2D, and as the previous algorithm it relies on the principal of dilation to form a Euclidean circle using several different structuring elements.
- *GPU based methods:*
  These explicit and implicit methods are usually not suited for 3D raster computations. Although interesting from an efficiency point of view, these methods are out of scope for this research, since the GPU does not fall within the scope.
- *Hierarchical Raster Method for Computing Voronoi Diagrams Based on Quadtrees:*
  This implicit method applied in 2D aims to increase the efficiency of normal dilation methods by using a quadtree. The quadtree should allow for quickly assigning an entire area a single value, instead of pixel-by-pixel. However, it is not clear how the quadtree should be implemented correctly when the Voronoi cells are not yet defined.
- *Using a KD-tree:*
  This explicit method employs a kd-tree to perform efficient nearest neighbour searches in 2D. This method is also possible to employ in 3D however, adding and removing seeds leads to an unbalanced tree. In order to perform nearest neighbour searches, the tree must be rebalanced after additions or removals.

The discussion of these methods will be used to develop an algorithm that is able to create a discrete 3D Voronoi diagram. This is done in Chapter 4.

# 4. CREATION AND IMPLEMENTATION OF THE DISCRETE 3D VORONOI DIAGRAM ALGORITHM

In this chapter a new algorithm for the creation of the discrete 3D Voronoi diagram will be presented. First of all, the process of how this algorithm is created will be described. This is quite important, since it explains certain characteristics of the algorithm, especially in light of the literature review in Chapter 3. The presented algorithm is based partly on Schueller's natural neighbour sweep circle algorithm.

After the algorithm is presented, some additions will be discussed that enable the algorithm to add, move and remove points within an already created discrete Voronoi diagram. In Chapter 2, several different functionalities have been mentioned that are deemed to be important for a GIS to have in order to handle the 3D discrete Voronoi diagram. The different GIS or other software packages that are capable of handling the discrete 3D Voronoi diagram have been identified, and their capabilities have been assessed. It is clear that some basic functionality is lacking. This functionality has also been added into the algorithm, and will be presented.

Finally, the possibilities of the algorithm will be discussed in terms of adaptation, future use and different environments (metrics)

In this chapter, the term voxel is used in all instances. Also in images that describe processes or concepts in 2D, the term voxel is still used in order to maintain consistent, and to emphasise that the algorithm is implemented in 3D.

## 4.1. HARDWARE, SOFTWARE

The entire project was programmed in the Python programming language (van Rossum, 2008), in combination with the Numpy module (Oliphant, 2006). The scripts were run on an AMD Athlon 64 X2 Dual Core 5000+ (2 cores at 2.60 GHz), with 3072 MB of RAM. The Python scripts were always executed on only one core, under Windows Vista Ultimate. Grass was run on the same system under Linux Ubuntu 8.10.

## 4.2. PRELIMINARY WORK IN AID OF CREATING THE ALGORITHM

The main algorithm, and extensions to it, were not just simply programmed in one go after the needs and method were defined. Actually these have changed even throughout the process of creating the algorithm. Therefore, the major steps in the evolution of the final algorithm will be discussed. Not in the least because this will clarify the literature discussion on discrete Voronoi diagrams even more.

As with the discussion on the discrete Voronoi diagram literature, the methods in this section will be described in 2D but can, without exception, be generalized to the 3$^{rd}$ dimension.

### 4.2.1. USING AN IMPLICIT METHOD

The first algorithm I created was based on the naïve method. This algorithm visits every voxel in the image, and checks for every voxel which seed is closest to that voxel. It does this by simply calculation the Euclidean distance between the voxel coordinates and the seed coordinates. As already mentioned, the naïve method is the most inefficient of all methods. For this reason it has been abandoned relatively quickly.

After this limited test, the choice was made to develop an implicit method, in order to create conceptually relatively simple algorithm that at the same time had potential to be relatively efficient. As explained in Section 3.3, an implicit method makes use of certain assumptions that enables the algorithm to assign a certain seed ID value to a voxel without having to perform distance calculations for every combination of voxel and seed. The choice to use an implicit method was made because the two explicit methods that were relatively promising in terms of efficiency are not easily implemented. These methods are:

1) *The methods created to work with the GPU.*
   Because the using the GPU is outside of the scope of this project, these methods will not be utilized in this thesis.
2) *The method based on kd-tree nearest neighbour search.*
   This method is very promising indeed. The kd-tree provides an excellent basis for nearest neighbour search. The main reason why this option has not been pursued is a very practical one. Finishing this master thesis within a reasonable time period relied partly on creating a program that would be technically and conceptually not too complicated. After an inventory of available programming libraries was made, it became clear that there is no library that enables me to create a kd-tree in three dimensions and at the same time can be easily integrated with tools and programs that I use. Creating such a library is quite arduous and would be far too time-consuming. Although this means that the method is not appropriate for this research, it should not be forgotten in future research.

Since no explicit methods were readily available to implement in the tool setup that was available, the only option is to proceed with an implicit method.

### 4.2.2. STRUCTURING ELEMENTS

The first attempt at creating a implicit algorithm is based partly on the Nearest Neighbour Sweep Circle algorithm, explained in Section 3.3.2 (Schueller, 2007). This method is in turn based on dilating seeds by several structuring elements in such a way that at the end of each round, the area around each seed represents an Euclidean circle as best as possible. Calculating which structuring element must be placed where is time-consuming. The increase in efficiency is based solely on the fact that voxels can be assigned to a certain seed without having to check if it was assigned correctly, as long as those voxels

are not near the boundary of a Voronoi cell. The voxels near the boundaries have to be evaluated nonetheless.

### Circle dilation

The first improvement I attempted to make was to avoid the calculations that were necessary to find the correct structuring elements for the right place. This would be possible if a circle could be created in a different way. Unfortunately, creating circles in a raster environment is very time-consuming. And although Bresenham created a well known algorithm for the creating of circles (Bresenham, 1977), it is still not efficient enough to improve the Nearest Neighbour Sweep Circle. Also, when dilating circles in other manners, artefacts can be seen (voxels that have not been assigned a value) (Ji, Piper et al., 1989) (see Figure 4.1).

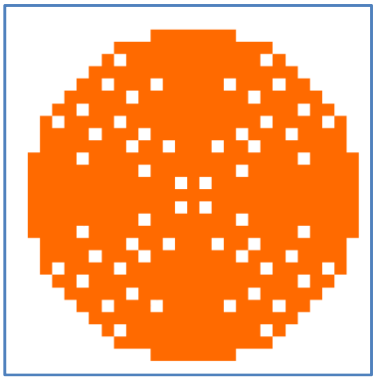


**Figure** 4.1 Result of an expanding circle using the Bresenham method

Other structuring elements

Another method to remove the time consuming checks to approximate a Euclidean circle is to simply use only one structuring element. Since there is no way to dilate a circle with one structuring element (Ji, Piper et al., 1989), this will eliminate the need for those calculations. However, this will also partly change the assumption that voxels that fall within the dilated area are assigned correctly (as was the case with the circular shaped dilation). This can be explained as follows:

The image in Figure 4.2a is a small raster (10x7 voxels) in which two seeds have been inserted. These seeds are iteratively dilated with a Moore neighbourhood structuring element, resulting in expanding squares. Figures 4.2b-d represent the image after respectively 1,2 and 3 rounds. A problem that can occur with this dilation is the wrongful claim of a voxel.

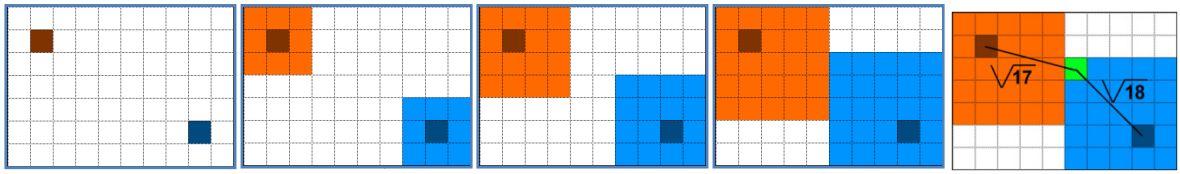



**Figure** 4.2 Dilation of two seeds **(a)**-**(d)**, and the possible incorrect claiming of a voxel **(e)**

The green voxel in Figure 4.2e shows a situation that could introduce errors. In the last round, this has been claimed by the blue seed. But when looking purely at Euclidean distance, the distance to the blue seed is $\sqrt{18}$, while the distance to the orange seed is $\sqrt{17}$. Therefore, this voxel has been wrongfully assigned to the blue seed. The way to overcome this is to allow the algorithm to re-assign voxels that have already been claimed. How this is done will be covered in detail in the next section.

A different structuring element that can be used is the diamond shaped structuring element, called the Neumann Neighbourhood. In the article by Schueller, 2007, this neighbourhood was also used, and it yielded both the best accuracy and the best efficiency. Note that the error problem illustrated in Figure 4.2 is not related to the shape neighbourhood; the same error can occur using the Neumann neighbourhood.

Another structuring element will also be tested, namely the 16-connected neighbourhood set presented in van Bemmelen, Quak et al, (1993). This neighbourhood is illustrated in Figure 4.3.

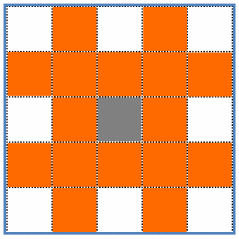


**Figure** 4.3 The 16-connected neighbourhood.

It is used by van Bemmelen , Qauk et al. to increase accuracy in finding a least cost path in a raster where different pixel values represent different costs. In this article there are more neighbourhood sets presented like the 16-connected, but larger (32-connected neighbourhood). If testing shows promising results, these will also be evaluated.

In the testing of this algorithm, the above three mentioned structuring elements will be used and compared in Section 5.3.2 (Accuracy and precision).

This method described above, and illustrated in Figure 4.2, is similar to that of Schueller's natural neighbour sweep algorithm, in that it uses a form of dilation. However, where the sweep algorithm would check every pixel if it falls within the expanding circle, this method implicitly assumes that the pixels that are claimed by the dilated cells are claimed correctly. Only if there is a collision (where two expanding cells try to claim one pixel) are explicit checks made to see to which seed a pixel belongs. This method might be more efficient, and is also conceptually quite simple. Therefore, the final algorithm will use dilation with one structuring element as its primary mechanism. The description of the algorithm is presented in the next section.

## 4.3. THE CREATEDISCRETEVD ALGORITHM

In the previous sections, several methods of creating a discrete 3D Voronoi diagram have been presented and discussed. The reason for choosing an implicit dilation algorithm has also been explained. In this section, the actual design and implementation of the algorithm will be presented.

As already mentioned, the algorithm will be explained and presented in 2D, but every aspect of it can be generalized in to 3D. So although the figures are in 2D, the algorithm is fully implemented in 3D.

### 4.3.1. THE GENERAL FORM

The goal is to make an algorithm that is able to dilate seeds in such a way that the result will be a true Voronoi diagram. To do this the algorithm will have to exist out of three basic components:

1) A loop that will iterate until the Voronoi diagram is complete.
2) A mechanism that will dilate seeds in a constant rate

3) Since the structuring element is not a circle, a mechanism must be in place to ensure **all** voxels have been assigned correctly once the algorithm has completed.

The algorithm `CreateDiscreteVD` presented in pseudo code below adheres to all the above criteria.

Algorithm 1: `CreateDiscreteVD`

```
1     change = 1
2     DO WHILE change = 1
3           FOR each voxel X in image:
4                 IF X == 0 THEN:
5                       PASS
6                 neighbourList = list of all neighbours of X
7                 FOR each neighbor N in neighbourList:
8                       IF N.id == X.id THEN:
9                             PASS
10                ELSE IF N.id == NoData THEN:
11                      N.id = X.id
12                ELSE IF N.id <> X.id and N.id <> NoData:
13                      D1 = distance between N and seed with id: N.id
14                      D2 = distance between N and seed with id: X.id
15                      IF D1 < D2 THEN:
16                            PASS
17                      ELSE IF D2 < D1 THEN:
18                            N.id = X.id
19                      ELSE IF D2 == D1 THEN:
20                            Set value of N.id according to ARBITRARY_RULE
21                NEXT
22          NEXT
23          IF no voxel were changed THEN change = 0
24    LOOP
```

In order to create complete Voronoi diagram, this loop must be repeated until none of the voxels have changed their ID value. A bit more explanation is perhaps in order. It starts at line 1, with the beginning of a loop. It will loop over all voxels in the image. At every voxel it encounters, it first checks if it has been assigned a value. If not, this voxel will be skipped, and the next voxel is evaluated. If it has been assigned a value the algorithm proceeds by identifying all the neighbours of the voxel and storing them in a list.

This action is determined by the shape of the applied structuring element. If the Neumann neighbourhood is used, only the four direct neighbours are selected (top, bottom, left and right). If the Moore neighbourhood is used, the four diagonal neighbours are also selected.

Now another loop is created. This loop iterates over the selected neighbours. If the neighbour has the same ID value as the current voxel (X), it is correctly assigned already (LINE 8 and 9). If a neighbour has no value set (NoData), it automatically assumes the ID value of the current voxel (X) (LINE 10 and 11). If a value has been set, but not the same as that of the current voxel, some checks have to be performed (LINE 12 – 20). At this instance, 2 dilating regions have met at this voxel. In order to see if the voxel was claimed correctly by the other region, the distance between the neighbour N and the seed corresponding to voxel X, and the distance between neighbour N and the seed corresponding to neighbour N have to be checked. If one distance is smaller than the other, the corresponding seed ID value is assigned to neighbour N. If the two distances are exactly the same, a choice must be made, according to the so-called `ARBITRARY_RULE`. This rule can be based on several things, such as ID value, position, etc. One could also choose to flag such voxels as being boundary voxels. Which method is most suitable for what situation is explained in Section 5.3, but the standard in this thesis is that the `ARBITRARY_RULE` states that in conflict situations the pixel is assigned to the seed with the lowest ID (of the seeds that claims the pixel, not of all seeds).

It must be noted that to increase efficiency, the distance checks that are performed are actually based on the squared distance, to avoid the use of square root operations (which can be computationally expensive).

What the `CreateDiscreteVD` algorithm does is nothing more than to take the structuring element, place it over every voxel in the image that has been given a ID value already, and 'stamp' the structuring element onto the original image.

### 4.3.2. IMPROVING THE ALGORITHM

In reality the CreateDiscreteVD algorithm is not yet very efficient, since it has to pass over every voxel in the image, even empty voxels. Although these are instantly recognized by the program and the voxel is skipped, it still takes up unnecessary time. It also tries to dilate voxels that are situated far within dilated areas, which is quite unnecessary. The only voxels that need to be dilated are, in fact, the voxels at the outer boundaries of each dilating area. This criterion can be narrowed down even further by stating that only those voxels need to be dilated that are at the sections of the boundaries that are expanding. This also removes the need for dilating those voxels that are at boundaries that are stationary, either at the edges of the image or at the stable boundaries between areas. In effect, this means that only the voxels that –in the current round- have been assigned a different ID value than they originally carried (including no value or 0) need to be considered for dilation. This is easily done by storing each voxel that is being changed in a list (changedVoxelsList), which stores unique voxels only. To iterate over this list of voxels, line 3 will have to change into:

```
3    FOR each voxel X in changedVoxelList:
```

One other way to make this algorithm a lot more efficient is making sure that no voxel is evaluated twice. If no mechanism for this would be in place, this would occur very frequent. Consider the example shown in Figure 4.4

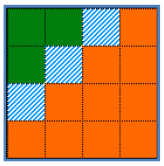


**Figure** 4.4 Empty border voxels that are examined twice

The orange voxels represent the boundary of a Voronoi area, during the process of dilation. When identifying the neighbours of the voxels, all the blue voxels are identified twice; once for the orange voxel below them, and once for the orange voxel to their right. In order to counteract this, a dictionary is maintained that stores only unique neighbours, thereby forcing the algorithm to use evaluate every neighbour once. However, this dictionary can store a neighbour twice, or more. This occurs when the voxel is a neighbour to two different voxels. In Figure 4.4 the blue voxels are a neighbour to both the green and the orange Voronoi areas. Both areas need to be allowed to claim the blue voxels. Therefore, not only are the neighbours stored uniquely with respect to their coordinates, but also in combination with the voxels they are identified from. Thus, all blue voxels will be stored in the dictionary as neighbour to the orange and green Voronoi area.

### 4.3.3. WHY THE ALGORITHM WORKS

As previously mentioned, an implicit method has to have a control mechanism that ensures that pixels are assigned a valid value. If a pixel would be assigned incorrectly, and there is no mechanism to check it, the erroneous value might progress through the Voronoi diagram. In the case of the `CreateDiscreteVD`, the control mechanism does not prevent pixels to be wrongfully claimed, however, it will ensure that pixels that have an invalid ID value are corrected. In other words, the key element that makes the `CreateDiscreteVD` algorithm work is the fact that voxels that have already been claimed can be overwritten, are overwritten in case they contain an invalid ID value. This is expressed in lines 12 through 20 of the pseudo code, and can be illustrated using in Figure 4.5.

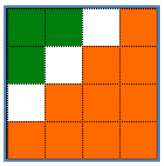


**Figure** 4.5 The boundary of two Voronoi cells.

This figure shows a voxel-level representation of two Voronoi areas, green and orange. The white voxels will be evaluated and claimed. The order in which the white voxels will be claimed is purely defined by the order in which the program loops through the **`changedVoxelList`**, which is usually on ID value. This means that one area will claim these voxels before the other has the opportunity to do so. Line 12 in the pseudo code represents the mechanism that enables the correct area to claim the voxel, even if it comes second in evaluating:

```
12          ELSE IF N.id <> X.id and N.id <>
```

This expression checks the condition that the neighbour **N** of voxel **X** does not have the same ID value as voxel X (which would make it part of the same area already) and that the neighbour N <> NoData. If this condition is met, this means that it has been claimed by a different area, and the algorithm will perform more checks. First of all, the (squared) distance between the evaluated voxel (**N**) and the seeds corresponding to both areas (green and orange in this case) will be calculated. If it is closer to the green seed, it will be assigned to the green area, and the same for the orange seed. If for this voxel nothing changes (i.e. it was already correctly claimed) nothing happens, and the seed is **not** added to the future `changedVoxelList`. However, if the voxel is changed, and therefore it was wrongfully claimed, it will be added to the future `changedVoxelList`. This behaviour makes it possible for every voxel to be assigned to the right seed, as long as the seed is represented by at least one voxel. This can be proven since:

## **Proof I:** All voxels will receive correct ID's

1) Any voxel (**X**) can claim a neighbouring empty voxel (**N**).
2) A voxel (**X**) can claim any already claimed neighbour (**N**), as long as:
    a. The seed **(a)** corresponding to (**X**) is closer to (**N**) than the seed **(b)** currently corresponding to **(N)**

    **OR** if (**N**) is equidistant from both **(a)** and **(b)**:

    b. The seed corresponding to the voxel (**X**) is favourable, based on the `ARBITRARY_RULE`.

This means in essence that a seed can propagate to every voxel the image, as long as the voxels it encounters are either empty, or if the voxels can be rightfully claimed for that seed based on either the distance or the `ARBITRARY_RULE`.

In order for the algorithm to perform correctly, it must also terminate every time when the Voronoi diagram is completed. That it does terminate in the correct way follows from the following proof:

**Proof II:** Correct algorithm temination.

1) Every voxel will receive the correct ID (**Proof I**)
2) If a voxel has been assigned the correct ID, it will not be assigned that ID again if it should be checked (line 15 of algorithm).
3) If all voxels have been assigned the correct ID, no voxels will change their ID
4) If no voxels have been changed in one round, the algorithm terminates (line 23 of algorithm).

#### 4.3.4. IMPLEMENTING THE BASIC ALGORITHM

According to the above design, the CreateDiscreteVD algorithm was coded and implemented. The algorithm was first tested in 2D, after which the algorithm was fully implemented in 3D. In order to test the algorithm, several test cases were created. Some were regular test cases (such as squares with seeds at the corners and/or midpoints) of which the outcome could be easily predicted and checked. Also some test cases were images containing randomly placed seeds. After the test phase of the basic algorithm was complete, more functionality was added.

### 4.4. ADDING BASIC FUNCTIONALITY

The previous section explained how the algorithm was designed and implanted. The result was an algorithm that produced the expected results. However, this initial implementation lacked basic functions like importing data. In this section, how the data is imported into a usable format is explained.

#### 4.4.1. IMPORTING DATA

The data that has been selected for this project was available in general text files (ASCII format). The files contained point data sets in which every sample point (future seed) is stored in a line of text (see Appendix I). Importing is done in a straightforward line-by-line fashion. Each line in the file is processed, and for each imported line (representing a point in the dataset) an instance of the Point class is created. This class has the following basic properties (more properties will be added to this class later):

| Name | Description | Type |
|---|---|---|
| Point.x | X-coordinate | `DOUBLE` |
| Point.y | Y-coordinate | `DOUBLE` |
| Point.z | Z-coordinate | `DOUBLE` |
| Point.a | Attribute value | `DOUBLE` |
| Point.id | ID value (unique) | `SIGNED INTEGER` |

**Table** 4.1 Properties of the `Point` class

Every newly created instance is added to the end of a list, called `pointList`. After the entire data file has been processed, `pointList` contains every point in the dataset.

### 4.4.2. SCALE

After importing the point data set, the points need to be converted into seeds, by entering them into the raster. Therefore, first the raster has to be created and to do this the extent of the raster has to be known. First of all, the extent is automatically created by placing a bounding box (rectangle) that exactly envelopes all points. The points are then re-referenced with respect to the extent so that the corner of the bounding box of all points with the lowest x, y, and z coordinates is assigned the raster index (0,0,0). These new coordinate values are then also stored into the point class:

| Name | Description | Type |
|---|---|---|
| Point.cropped_x | X-coordinate of point with respect to the origin of the bounding box | DOUBLE |
| Point.cropped_y | Y-coordinate of point with respect to the origin of the bounding box | DOUBLE |
| Point.cropped_z | Z-coordinate of point with respect to the origin of the bounding box | DOUBLE |

**Table** 4.2 Extended properties of the `Point` class: coordinates cropped to the bounding box.

In a raster, voxels are assigned an (i; j; k) index, instead of (x, y, z) coordinates for exact points. The index of a voxel is expressed in integers. In order to place the points into a grid as seeds and compute the i, j and k indices of the seeds, a resolution has to be decided on. This can be illustrated by the following relation between resolution and size of the bounding box (can be generalized for x-, y- and z – directions):

$$\text{Resolution (along x} - \text{axis)} = \frac{\text{length of boundingbox along x} - \text{axis}}{\text{number of voxels along x} - \text{axis}}$$

In order to create the grid, ultimately the number of voxels is needed. Therefore the user can enter either the resolution, or the number of voxels. The resolution is then used to scale the points into the grid by multiplying the cropped x ,y and z coordinates by the resolution. This gives the following additions to the Point class:

| Name | Description | Type |
|---|---|---|
| Point.scaled_x | X-coordinate of point scaled to the grid size | `DOUBLE` |
| Point.scaled_y | Y-coordinate of point scaled to the grid size | `DOUBLE` |
| Point.scaled_z | Z-coordinate of point scaled to the grid size | `DOUBLE` |

**Table** 4.3 Extended properties of the `Point` class: coordinates scaled to the bounding box.

A grid point (voxel) could be seen in exact form as a square of 1 x 1. In order to put exact points into a grid, the voxel that best matches that point has to be found, and marked by assigning it a certain value. The pixels are defined so that there is no boundary-line between them, so for instance, the extent of pixel *P*( k; l; m) is as follows:

$$k < P_{i=k} <= k + 1$$
$$l < P_{j=l} <= l + 1$$
$$m < P_{i=m} <= m + 1$$

This, combined with the fact that an exact point has no size, means that such a point will always fall within the boundaries of a voxel, and never on top of a boundary. By taking the scaled (exact) point, and converting it to an integer, one can find the voxel index. Consider Figure 4.6.

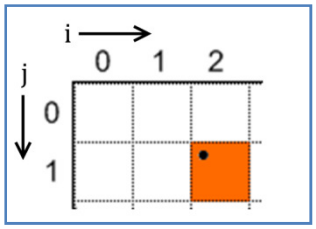



**Figure** 4.6 Converting exact points (black dot) to discrete coordinates (orange voxel).

In this figure, the exact scaled point with coordinates (2.2; 1.2) is displayed as a black dot. By converting these coordinates into integers, (2; 1) the (orange) voxel which represents this point is found. This operation yields another property for the Point class:

| Name | Description | Type |
|---|---|---|
| Point.grid_i | i-index of point scaled to the grid size | UNSIGNED INTEGER |
| Point.grid_j | j-index of point scaled to the grid size | UNSIGNED INTEGER |
| Point.grid_k | z-index of point scaled to the grid size | UNSIGNED INTEGER |

**Table** 4.4 Extended properties of the `Point` class: exact points converted to discrete points.

### 4.4.3. TRUE EUCLIDEAN DISTANCE

By converting exact, scaled points into voxels, the precision is automatically reduced. To make sure that this loss in precision does not yield more errors in the Voronoi diagram, some precautions must be taken. Suppose a Euclidean distance calculation is made between the centres of a voxel (1; 1) and a seed (2 , 2). Normally the outcome of this calculation would be $\sqrt{2}$. Now suppose the scaled point that is represented by this seed has the following coordinates: (2.9, 2.9). This would bring the true scaled distance to $\sqrt{3.61}$, which is a significant difference. This might have serious effects on evaluations of voxels that seem to have 2 equidistant seeds. In order to prevent this from occurring, the scaled coordinates of seeds will always be used in distance calculations.

### 4.4.4. COMBINING POINTS

Another issue that comes from the scaling of exact points into a grid is that of two or more points that fall within the boundaries of one voxel. This is quite possible, and is handled in the following way.

If during the process of scaling points fall within the same voxel, a new point is create with a negative ID value to indicate it is a combined point. This point has the extra property:

| Name | Description | Type |
|---|---|---|
| Point.combined_ids | All the ids of points that have been integrated into this point | SIGNED INTEGER |

**Table** 4.5 Extended properties of the `Point` class: combining unique points and their ID values.

The ID value of this point is the id value of the first point it stores multiplied by -1 to indicate that it is a compound point. The x, y, z and a values are averaged over all combined points which also results in averaged cropped and scaled values. Thus the two exact points:

Point (1.1, 1.1, 1.1) with attribute value 5 and ID =1
Point (1.7, 1.7, 1.7) with attribute value 10 and ID = 2

Would be combined to the following exact point:

Point (1.4, 1.4, 1.4) with attribute value 7.5 and ID = -1

Which would then be treated as a 'normal' exact point; it will be converted to a discrete point.

The attribute values of the original points are set to -9999, indicating that these need not be used anymore. The final result is one point that can be inserted into the grid, but which represents all the points that fall within the voxel.

### 4.4.5. DATA STRUCTURE

To store the discrete 3D Voronoi diagram, I propose to use the following data structure. In the next section, some extensions will be proposed. Necessary adaptations to the file formats to accommodate for these extensions will be shown in Section 5.2.

- `discrete_VD.a3d`
  This file contains the (3D) raster data. The format is the ASCII 3D format, used by GRASS GIS. For more detail on the internal file structure, see Appendix I
- `id_2_values.pnt`
  This file contains the attribute values that correspond to the ID values of the seeds. This file is also plain ASCII text, and formatted according to Appendix I.

## 4.5. EXTENDING FUNCTIONALITIES

After a discrete 3D Voronoi diagram has been created, it is possible that the Voronoi diagram has to be modified by adding removing or moving points. This is something that is particularly important for users that need interactive modelling functionalities (Bailey and Gatrell, 1995). It may be that some points need to be removed because they are regarded as outliers. Also, it might be necessary to insert extra data points (for instance, when new measurements become available), or to move data points to see what the effect is on the Voronoi diagram. To accommodate for these requirements, the program was extended with algorithms that provide functionality to add, move and remove points. These algorithms are presented in this section.

### 4.5.1. ADDING POINTS

The same concept that enables a seed to propagate throughout a grid to all 'correct' positions (Proof I) can be readily used to insert a new seed.

Algorithm 2: `AddSeed`

```
1          discreteVD[newSeed.x, newSeed.y, newSeed.z] = newSeed.ID
2          newDiscreteVD = CreateDiscreteVD(newSeed)
```

To do this, first of all, the voxel representing the new point would be assigned the ID value of the new point, thus becoming the new seed in the grid (line 1).

Normally, the original `CreateDiscreteVD` algorithm would each round rely on a list of points that were changed in the last round. This can be emulated by feeding this function a list containing only the new point (line 2). This would set the `CreateDiscreteVD` algorithm to evaluate the neighbours of this point. Assuming that the discrete Voronoi diagram was complete, these voxels will already be occupied, but as long as they should belong to the new seed, these voxels will be reclaimed by the new seed. In this way, the new 3D Voronoi diagram will be finished in as many rounds as it takes to dilate the new seed.

This method will not only allow for the insertion of one point, but can be used to simultaneously insert multiple new points. Building the adapted Voronoi diagram will then take only as many rounds as are necessary for the slowest Voronoi cell to be dilated.

### 4.5.2. REMOVING POINTS

In vector space, removing a point is a very difficult procedure in certain degenerate cases (Devillers and Teillaud, 2003; Ledoux, 2006). In raster environment, removing a point is relatively straightforward, and there are no degenerate cases. This makes the raster environment ideal for manipulation of data points.

The pseudo code for this algorithm is as follows:

Algorithm 3: RemoveSeed

```
1          listOfPixels = IdentifyPixels(ID_to_be_removed, discreteVD)
2          FOR each pixel P in listOfVoxels:
3                 FOR each neighbour N of P:
4                       IF N == ID_Value THEN:
5                             listOfBoundaryVoxels += [N]
6                       END IF
7                 END FOR
8          END FOR
9          newVD = CreateDiscreteVD( listOfBoundaryVoxels, discreteVD)
```

First, all voxels containing the ID that corresponds to the seed (line 1) (green voxels in Figure 4.6a) need to be identified. There are two ways to do this. The first is a quite inefficient operation, since it requires the program to evaluate each voxel in the image and check if it has the ID value that corresponds with that of the seed to be deleted. Fortunately, there is a module available for Python called Numpy (Oliphant, 2006), which is programmed in C, and provides very fast array processing operations. It has a built-in function that takes a 2D or 3D array together with a value, and returns the indices of all voxels that contain that value. This function performs the operation many times faster than a user defined function can do by evaluating every pixel. After the voxel indices have been identified, these voxels are visited to see if their neighbours belong to another Voronoi cell (line 4); if so, they are stored in a list (line 5) (see Figure 4.7 b).

The second possibility to get a list of all voxels that need to be deleted is to start with the seed that is to be deleted. Using the dilation principal, all neighbours are checked if they have the same ID value. If a neighbour does have the same ID value, its neighbours are checked. When a neighbour does not have the same ID value, it can be considered to be a 'boundary' voxel. This way all boundary pixels can be stored in a list and used in the next section of the algorithm. In theory, this is much faster, since it has to visit only the voxels that belong to the Voronoi cell that is to be removed, and its boundary voxels. However, in practice it turns out that is slower than the low-level pre-programmed method described above. Therefore, this method is not implemented. However, when implementing the algorithm in a different language, such as C, this issue must be revisited.

The second part of the removing procedure is performed by passing the list of boundary pixels on to the general dilation function (line 9), which will then 're-dilate' the Voronoi cells surrounding the removed Voronoi cell, overwriting the voxels that are to be deleted , finally yielding a recreated Voronoi diagram (Figure 4.7c).

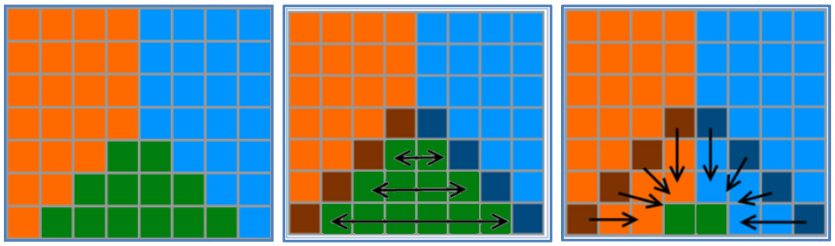



**Figure** 4.7 The process of removing a seed and its Voronoi cell.
**(a)** The Voronoi cell. **(b)** Identifying the green voxels and boundary voxels. **(c)** The darker coloured boundary voxels are then re-dilated until all green voxels are overwritten.

Identifying the necessary voxels is a very time-consuming task. However, there is another method which could not be implemented because of time restrictions. Because it makes use of certain concepts such as storing edges of Voronoi cells, which are yet to be presented, it will be explained in Section 5.2.

### 4.5.3. MOVING POINTS

The simplest concept that accommodates for moving points within an already created Voronoi diagram is to remove the point that is to be moved, and re-insert it at a different location. Since both operations are already available, they are combined into one function. Running this function will first call the RemovePoint function then add a point in the new location with the AddPoint function, resulting in a modified Voronoi diagram.

These additional algorithms that allow to add, remove and move points, also show one of the strong points of the `CreateDiscreteVD` algorithm in comparison to other discrete 3D Voronoi diagram algorithms, because in essence the AddSeed algorithm uses the `CreateDiscreteVD` algorithm to 'overwrite' certain pixels, so no major adaptations are necessary. The modifications are also done locally, instead of needing to reconstruct the entire Voronoi diagram. Also the RemoveSeed algorithm shows similar efficiency if an additional data structure is added, which is described in Section 5.2

## 4.6. ADDING LACKING FUNCTIONALITIES

In the chapter concerning the state of the art in 3D (raster) GIS, the functionalities that the evaluated GIS offers were identified, and compared with the functionality that was desirable for combining these GIS with the discrete 3D Voronoi diagram. Two operations were not available that would both increase the possibilities of the discrete 3D Voronoi diagram quite drastically, resampling and natural neighbour interpolation. The creation and implementation of both operations will be presented and discussed.

### 4.6.1. RESAMPLING

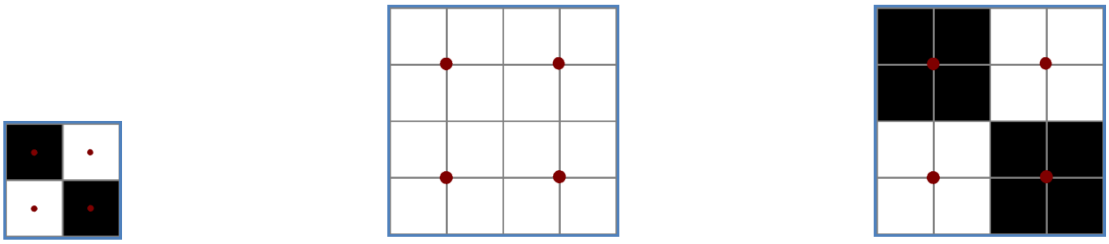
By resampling an image, one can increase or decrease the spatial resolution of that image. It is also used in raster rotation and (projection) transformations. By upsampling an image, the number of voxels is increased, resulting in more voxels covering the same image. By doing so, the spatial resolution of an

image increased. Downsampling on the other hand, will result in a lower spatial resolution, by decreasing the number of voxels (Baxes, 1994).

Upsampling

Upsampling can for instance be used to match the resolution of two different raster images. If they have different resolutions, the image with the lowest spatial resolution can be upsampled to match the resolution of the second image. This can be necessary when performing an operation like map algebra.

Although upsampling will yield a higher spatial resolution, the features in the image will only be enlarged; they will not become more sharply defined or more detailed, since the upsampling cannot retrieve more information out of the original image than there already is. Essentially, upsampling is nothing more than interpolation of an image, in this case using nearest neighbour interpolation. This can be illustrated using Figure 4.8.



**(a)** **(b)** **(c)**

**Figure** 4.8 Resampling of a 2x2 raster **(a)** to a 4x4 raster **(b)** and **(c)** using the nearest neighbour approach.

In Figure 4.8a , a 2x2 image is shown. This has to be resampled, in such a way that the new image is 4x4 Figure 4.8b. Each centre of a voxel in **(a)** can be represented in **(b)** as a point, which has the same relative position in **(b)** as it has in **(a)**. By performing a nearest neighbour interpolation on the voxels with respect to the points, one can get the resampled image in Figure 4.8d.

When a digital image is upsampled, for example a digital photograph, using a nearest neighbour interpolation will cause aliasing (Baxes 1994). Aliased images are images in which the spatial pattern is distorted, resulting in a less clear image. Therefore, other interpolation methods, such as bi-linear, tri-linear or bicubic interpolation are usually implemented (Baxes 1994). However, these result in an image that contains intermediate values, so the result looks smoother (Baxes, 1994). For normal digital images this indeed preferable, but in this case the required result can only contain values identical to those in the original image, since intermediate values would result in Voronoi cells with fuzzy boundaries.

Since creating a Voronoi diagram is nothing more than interpolating points using nearest neighbour method, the program already contains a nearest neighbour interpolation method: the

`CreateDiscreteVD` algorithm. This can be used to upsample an already existing image by first creating a grid that has the required amount of voxels. Then each voxel from the original image can be represented as a seed in the new image. A list of these seeds is created, and passed onto the dilation function. The result is a resampled image with the desired resolution. This is theoretically similar to a standard nearest neighbour resampling algorithm (Baxes, 1994).

### Downsampling

The most likely reason for downsampling a discrete Voronoi diagram is to reduce the size of the Voronoi diagram so that certain analyses on that Voronoi diagram can be performed faster. If the operation does not require the level of detail provided in the original discrete Voronoi diagram, there is no objection to do so.

The concept of downsampling can be illustrated using Figure 4.9, in which a downsampling operation from a 3x3 image to a 2x2 image is performed.

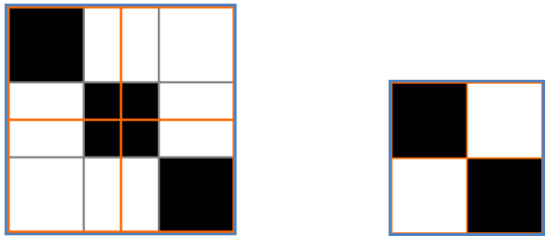


**Figure** 4.9 Downsampling from a 3x3 **(a)** to a 2x2 **(b)** voxel image.

In Figure 4.9 a, the grey lines represent the voxels in the 3x3 image. The orange lines define the areas that represent the voxels in the 2x2 image. Although the voxels have the same dimensions, conceptually one could say that a voxel in the 2x2 image overlaps multiple voxels in the 3x3 image. The value that is assigned to the voxels in the 2x2 image depends on the values of the voxels in the 3x3 image they overlap. These values can be averaged with respect to the area they envelop, but in the case of discrete 3D Voronoi diagrams, averaged values are not acceptable. This is because of the fact that the cells of a Voronoi diagram have a sharply defined boundary. Averaging values would make these boundaries fuzzy, and thus the Voronoi diagram would not be a true discrete Voronoi diagram anymore.

The `CreateDiscreteVD` algorithm assigns the value that has the most area within the area that represents a voxel in the 2x2 image. Consider the top-left voxel of the 2x2 image. This voxel envelopes two values in the 3x3 image; 0 (black) and 1 (white). The total area of the black voxels in the top left area is 1 + 0.25 = 1.25 voxel. The area of the white voxels is 0.5 + 0.5 = 1 voxels. Therefore, the top left voxel in the 2x2 image is assigned 0 (Figure 4.9b). In case the total areas of all distinct values are equal, one is chosen randomly.

This method has not been documented as far as I can find. However, this is not surprising, since it will yield quite poor results for digital images, such as photographs. This is because it will not average any values, similar to the upsampling mechanism.

In general, it can be stated that resampling a 3D raster image is computationally expensive.

### 4.6.2. NATURAL NEIGHBOUR INTERPOLATION

One direct possible use of the discrete 3D Voronoi diagram is to perform a natural neighbour interpolation on the dataset. The natural neighbour interpolation is a method that can be performed with the Voronoi diagram or the Delaunay triangulation as a basis (Sibson, 1980) and is used in several fields of science that make use of datasets containing scattered data such as engineering, computer sciences and geosciences (Gold, 1989; Gold, 1990; Perez and Traversoni, 1996; Boissonnat and Cazals, 2001; Boissonnat and Cazals, 2002; Yanalak, 2003; Yanalak, 2004). According to Perez and Traversoni (1996), the natural neighbour interpolation is rarely used, even while the resulting interpolated surfaces are smooth. The authors state that one of the reasons for this fact might be that the implementation is relatively difficult in 2D, and it can be expected that the 3D implementation of the natural neighbour interpolation is at least as difficult. In Boissonnat and Cazals (2002), an implementation of natural neighbour interpolation is presented and discussed. However, it has not been implemented yet in for instance the CGAL library (CGAL, 2009), which is a widely used computational library for the C++ programming language (and other languages). This is an indication that the implementation of the natural neighbour interpolation is not a simple task. In this section, it will be shown how the discrete 3D Voronoi diagram can be used to implement the natural neighbour interpolation.

In contrast to for instance the Inverse Distance Weighted interpolation, where each seed has an influence that is based on distance, the natural neighbour interpolation method is a natural neighbour interpolation method, which is based on overlapping Voronoi areas (2D) or volumes (3D). It uses a weighted average of neighbouring areas or volumes to determine the influence of each seed. In Figure 4.10a-d, the mechanism of assigning the weights for each is illustrated.

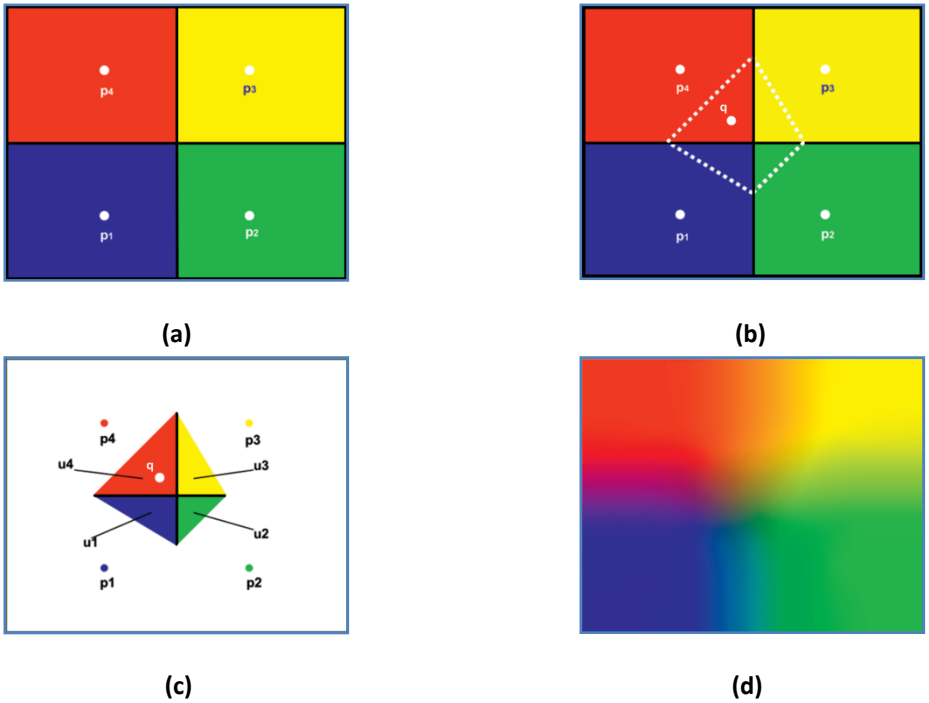



**Figure** 4.10. The basic principle of natural neighbour interpolation. **(a)** A Voronoi diagram of 4 regularly spaced points ($p_1$-$p_4$). **(b)** Insertion of point *q*, to be interpolated. Dotted white polygon shows the new Voronoi cell. **(c)** The new Voronoi cell, with $\mu_1$-$\mu_4$ representing the respective weights for each corresponding seed, based on the relative area of each section with respect to the area of the new Voronoi cell. **(d)** The interpolated image, after each point has been interpolated.

Figure 4.10a represents an exact Voronoi diagram of 4 points, $p_1$ to $p_4$. The areas are coloured for clarity, and represent different values. For each location that is to be interpolated (in this case, each voxel in the image) a new point is temporarily inserted (point q) and a new Voronoi diagram is created. The dotted line in Figure 4.9b shows the area of the newly created Voronoi cell. The overlapping areas ($u_1$ to $u_4$) that this new Voronoi cell has with the original Voronoi cells determine the relative weight of the seed that corresponds with those areas. The value of voxel q ( $a_q$) can be described as:

$$a_q = \frac{\sum_{i=m}^{n} u_i a_{p_i}}{\sum_{i=m}^{n} u_i} \qquad \text{Eq. 6}$$

It is important to note that the inserted Voronoi cell is only temporary. For each interpolated voxel, a new temporary Voronoi cell is created. After the value at this voxel estimated, the cell is removed. If this is done for every voxel in the image, the result is an interpolated raster, approximating the (continuous) field that was sampled.

The natural neighbour interpolation method has some interesting properties. These are explained thoroughly in Sambridge, Braun et al. (1995), Watson (1992) and Gold (1989)and will be summarized here.

Properties of the natural neighbour interpolation method

The first attractive feature of the natural neighbour interpolation method is that it is an exact interpolation method. This is in contrast with, for instance, the Inverse Distance Weighted interpolation method, which is an inexact method. The fact that it is an exact interpolation method implies that at the sample point location (or seed location, in the context of the `CreateDiscreteVD` algorithm) the interpolated value is exactly the same as the recorded value. In other words, the weight of each seed is exactly 1 at the exact location of that seed, which means that no other seeds influence the value at that location.

Secondly, the natural neighbour interpolation method is a local method, meaning that the influence of a seed is restricted to a certain area, so that a location is not influenced by points that are situated relatively far away.

The third property of the natural neighbour interpolation method is that the resulting surface is continuous, meaning differentiable, except at positions of the seeds. It must be noted that the natural neighbour interpolation method can be adjusted so that its derivative is existent everywhere ($C^1$), also at the seed locations (Watson, 1992).

### 4.6.3. IMPLEMENTING THE NATURAL NEIGHBOUR INTERPOLATION

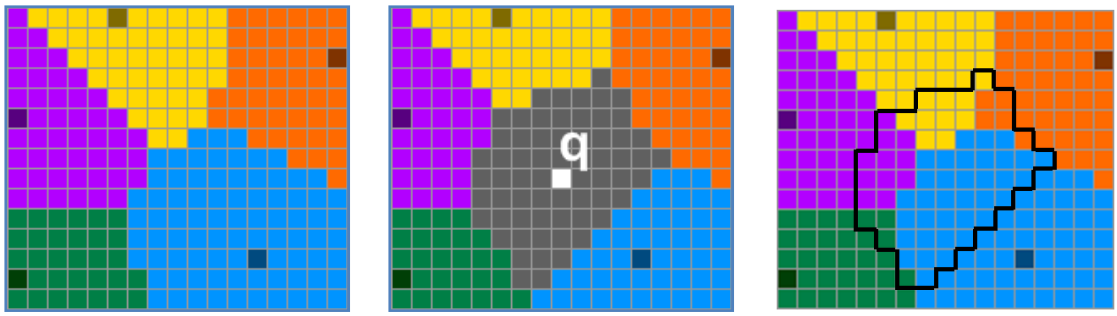
Because normally the Voronoi diagram is created in exact form -especially in GIS-, the exact form is usually also used to create the natural neighbour interpolation. The result of the natural neighbour interpolation method can be an entire image (a raster), but in theory it can also be a single point (if the user only wants to know the value at that point). In this research, the output will be a 3D raster of the entire dataset.

The output of the `CreateDiscreteVD` algorithm is a 3D Voronoi diagram in 3D raster format. Once the 3D discrete Voronoi diagram is created, this raster can also be used to as a support for the natural neighbour interpolation, instead of the exact 3D Voronoi diagram used normally. In Park, Linsen et al. (2006), this is done very efficiently. First of all because they create the discrete Voronoi diagram by using the kd-tree, which can be a very efficient method as already explained in Section 3.3. Secondly, they also use the power and properties of the GPU to effectively create the natural neighbour interpolation of the discrete Voronoi diagram. Although this is not possible in this project, the function to add a point in the discrete Voronoi diagram is relatively efficient, since only the voxels that are changed are evaluated, instead of the whole raster. This makes it very suitable for the natural neighbour interpolation. This is done in a similar way when using an exact Voronoi diagram as the basis for the natural neighbour interpolation, and is implemented as illustrated in the following pseudo-code:

Algorithm 4: `CreateNaturalNeighbour`

```
1          discreteVD_mod, modifiedPixelList = AddSeed(discreteVD, P)
2          FOR each pixel Q in modifiedPixelList:
3                totalValue = totalValue + discreteVD[Q.x; Q.y]
4          END FOR
5          interpolateValue = totalValue / numberOfModifiedPixels
```

In line 2 the AddSeed algorithm is called, which is introduced in Section 4.5. It takes the indices of the voxel (P) that is to be interpolated and the original discrete Voronoi diagram (Figure 4.11a) as parameters. It returns the modified Voronoi diagram (Figure 4.11b) containing the new Voronoi cell, and the list of voxels that were modified in the process. In line 4 of the pseudo code, the values of the voxels in the original Voronoi diagram (voxels within black line in Figure 4.11c) that correspond to the modified voxels (grey voxels Figure 4.11b) are summed. This total is then divided by the number of modified voxels. This gives the interpolated value at voxel P. To get the total interpolated image, this has to be done for each voxel in the 3D Voronoi diagram.




**(a)** **(b)** **(c)**

**Figure** 4.11. Interpolation a discrete Voronoi diagram using natural neighbour interpolation. **(a)** The discrete Voronoi diagram. **(b)** The inserted point *q,* to be interpolated, and the corresponding new Voronoi cell (grey area). **(c)** The amount of voxels within the black outline corresponding to a specific seed, relative to the total amount of voxels within the outline, determines the weight for each of the corresponding seeds.

In practice this means that the volumetric calculations that can be difficult to implement in exact environment (Ledoux and Gold, 2005), comes down to simply counting voxels, and multiplying the number of voxels by their (known) volume.

## 4.7. SUMMARY

In this chapter, the development and implementation of the `CreateDiscreteVD` has been discussed. The algorithm is based on the implicit Nearest Neighbour Sweep Circle algorithm by Schueller (Schueller, 2007), with several modifications in the dilation component to make it more efficient.

Besides the basic `CreateDiscreteVD` algorithm, 3D raster data import/export functionality, dynamic modelling functionality (adding, moving and removing points), resampling functionality and natural neighbour interpolation have been added.

The result is an algorithm that is able to import a 3D point dataset and from this dataset create a discrete3D Voronoi diagram. It can export this Voronoi diagram to different formats so that it can be imported in both MayaVi2 and GRASS. Besides the GRASS 3D raster functionality such as map algebra, it is possible to perform the neighbour interpolation on the discrete 3D, thereby creating a 3D continuous surface.

# 5. USING THE DISCRETE 3D VORONOI DIAGRAM TO MODEL CONTINUOUS INFORMATION IN GEOSCIENCES

Besides designing and implementing an algorithm for the creation of the discrete 3D Voronoi diagram, combining the discrete 3D Voronoi diagram with current off the shelf GIS is also something that has to be assessed. The software that enables the handling of discrete 3D data has been selected, and in this chapter, the usage of the discrete 3D Voronoi diagram in combination with trivariate (x, y, z, a) data will be assessed. How useful is the discrete 3D Voronoi diagram in practice? How does it compare to the exact Voronoi diagram, in terms of accuracy, speed and usability? These questions will be addressed in this chapter.

In Section 5.1, the 3D Voronoi diagram is assessed in the context of modelling continuous 3D data. In Section 5.2 other (future) possible uses and extensions are presented, that might increase the usefulness of the discrete 3D Voronoi diagram. Section 5.3 shows the advantages and disadvantages of the discrete 3D Voronoi diagram compared to those of the exact 3D Voronoi diagram. Finally, in Section 5.4 the use of the discrete 3D Voronoi diagram in a GIS environment is assessed and discussed.

## 5.1. USING THE 3D VORONOI DIAGRAM FOR THE MODELLING OF CONTINUOUS FIELD DATA

Basically there are two ways of using the discrete 3D Voronoi diagram for the modelling of continuous data:

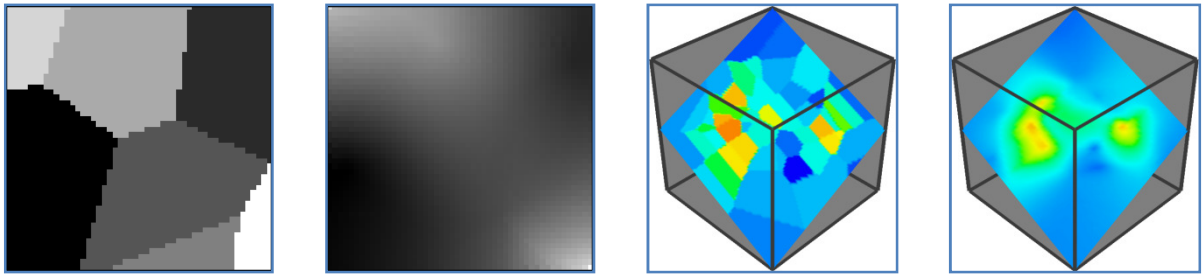
- Using the discrete Voronoi diagram as a basis for creating a continuous field using natural neighbour interpolation.
- Using the discrete Voronoi diagram as a tool for neighbourhood-related spatial analysis

Although interpolation could be considered to be a form spatial analysis, it is mentioned here separately, since it is a very valuable means of using the Voronoi diagram.

### 5.1.1. MODELLING CONTINUOUS FIELDS

The main research objective is: *To assess the use of the discrete 3D Voronoi diagram for the modelling of 3D continuous information in geosciences.* Since a model is always an approximation of reality, a viable question is then: How to approximate a (trivariate) continuous field with the help of the discrete 3D Voronoi diagram? For this, an interpolation scheme is necessary.

Interpolation the discrete 3D Voronoi diagram using natural neighbour interpolation is relatively easy: No user input concerning any parameters is required. After the algorithm is complete, the image is interpolated. In Figure 5.1 both a 2D and 3D discrete Voronoi diagram are shown, with the resulting interpolated images.



**(a)** **(b)** **(c)** **(d)**

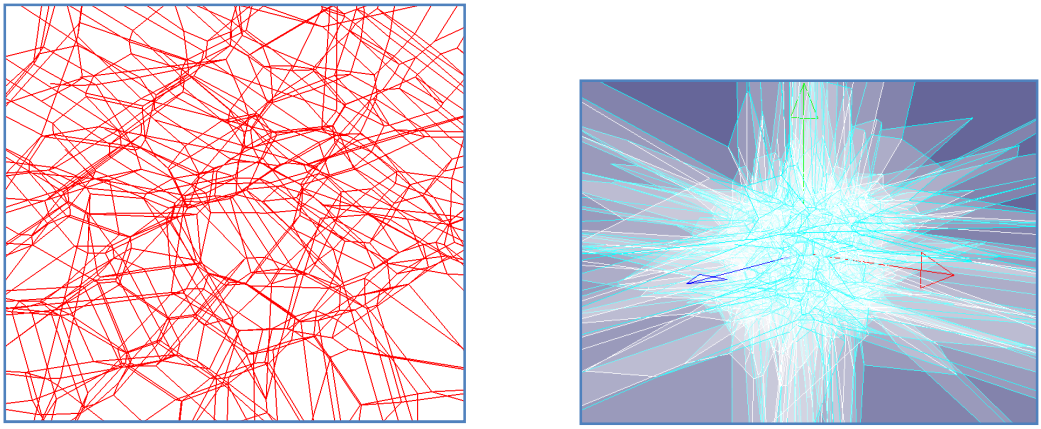
**Figure** 5.1. **(a)** A 2D discrete Voronoi diagram and **(b)** its interpolated surface. **(c)** Slicing of a discrete 3D Voronoi diagram and **(d)** a slicing of the interpolated surface.

As previously mentioned, the exact natural neighbour interpolation method is relatively difficult to implement, mainly due to degenerate cases of sample point locations. Since the discrete 3D Voronoi diagram does not have this problem, the implementation of such an interpolation method is relatively straightforward. The algorithm itself has already been presented. However, due to the speed limitations concerning Python, the algorithm is not yet practical to implement. At this moment, it takes 2 hours to interpolate a 50x50x50 Voronoi diagram with 150 seeds, whereas using the exact Voronoi diagram model by Ledoux (2006), it takes approximately 10 minutes. The implementation by Ledoux was programmed in a compiled language called Delphi.

### 5.1.2. Neighbourhood-related spatial operations and analysis

The discrete 3D Voronoi diagram itself can be a valuable resource of information, without having to modify, manipulate or analyze it. The structure itself can be employed to assess the data that has been used to create it. To clarify this, one of the useful properties of the Voronoi diagram will be revisited. As already mentioned, the Voronoi diagram is very well suited for handling data sample points that are anisotropically distributed, since the shape and size of a Voronoi polygon is dependent on the distribution of the points (Ledoux, 2006). Also, the distribution of sample points can be inferred more or less intuitively from the Voronoi diagram, simply by a visual inspection. In 2D, this is true for both the exact and discrete Voronoi diagram; when faced with an image of either of the two, one can rather easily point out where the seeds must be.

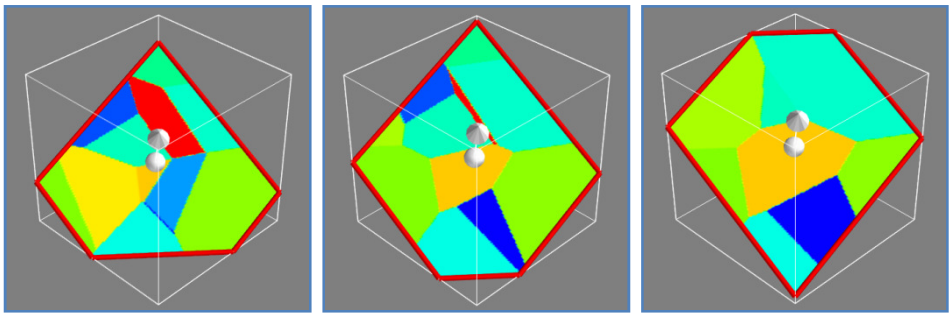
However, in 3D this is slightly different. In order to display the exact 3D Voronoi diagram, a wireframe can be used, as is done in Figure 5.2a, but unfortunately this does not clearly show the concept of depth. For this shading or surface visualizing is required, which has been done for a different Voronoi diagram in Figure 5.2b, but this makes it not much clearer.



**(a)** **(b)**

**Figure** 5.2. Visualization of an exact 3D Voronoi diagram can be a difficult task. **(a)** An exact 3D Voronoi diagram visualized using online lines and **(b)** displaying surfaces as well. Both are not easily interpreted.

A discrete Voronoi diagram is also difficult to visualize. Section 2.2 discussed the possible ways to visualize the discrete 3D Voronoi diagram; volume rendering, isosurfacing and slicing. Visualization of the entire Voronoi diagram through volume rendering is not recommended, since this will show every Voronoi cell in the raster. Isosurfacing is also not a recommended option for the raw discrete Voronoi diagram; this is used more appropriately for the processed and interpolated result. However, by slicing the data, the distribution of points can be illustrated very well. An example of this is show in Figure 5.3



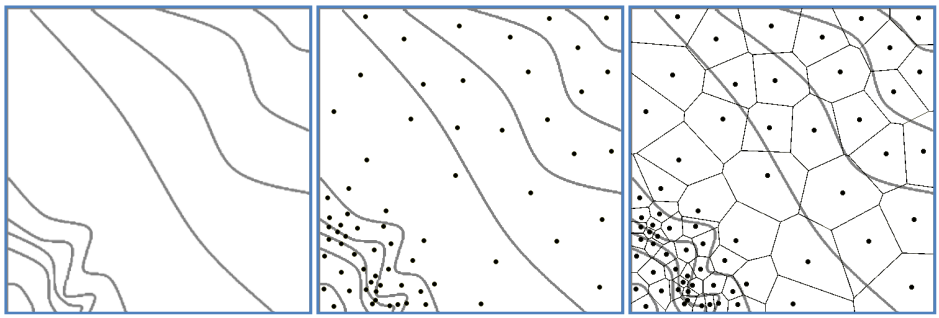
**(a)** **(b)** **(c)**

**Figure** 5.3. **(a)**-**(c)** Visualization of a discrete 3D Voronoi by using a special slicing tool in MayaVi2, allowing for 3D visualization control. Three consecutive slices of one 3D Voronoi diagram.

It shows an image of MayaVi2, in which a 3D discrete Voronoi diagram has been loaded and sliced by a specified plane. By simple mouse movements, this plane can be controlled in x-, y- and z-directions. This way, the distribution of points in all directions can be visualized and interpreted. It must be noted here that the resulting slices do not represent 2D Voronoi diagram, but mere a cross-section of 3D Voronoi diagrams. The same goes for all other images in this thesis that show slicing of a 3D Voronoi diagram.

It might seem trivial to look at the distribution of sample points. However, if the user only has the discrete Voronoi diagram, and not the dataset of 3D points, it can be quite important to look at the sampling distribution. An example: suppose a method called 'progressive sampling' is used during the collection of data. In this method, the data that is sampled is analyzed simultaneously. In places where changes in the sampled variable are minor, it can be decided to collect fewer samples (Burrough and McDonnel, 2006). This method can decrease the amount of necessary sample points and calculation time, but it will also increase the anisotropy of the sample point distribution.

When this sampling method is used, the discrete Voronoi diagram based on these data will contain large cells where the change in the measured variable is relatively low. In areas where this change is larger, the Voronoi cells will become smaller. An example of this can be seen in Figure 5.4 a-c:



**(a)** **(b)** **(c)**

**Figure** 5.4. Using the Voronoi diagram as a means of interpreting data and data quality, by analysing the distribution of Voronoi cells and seeds in a progressive sampling schema. **(a)** A hypothetical relief map showing contour lines. **(b)** Possible sample point distribution. **(c)** The generated Voronoi diagram. Small Voronoi cells indicate high sampling density, and high change in measured attribute value.

If a user would receive such a discrete Voronoi diagram, and it is known to him that progressive sampling has been used, he can make an interpretation of the data simply by looking at the Voronoi diagram. It is important to note however, that the shape and size of the cells say little on the modelled value itself, but only on the change in value; since in progressive sampling schemes the points are closer together wherever the sampled values change a lot. In the case of Figure 5.4 this attribute would be elevation.

Using the Voronoi diagram to assess data and the distribution of sample points is just one of many ways in which the Voronoi diagram can be used to perform spatial analysis. Many more spatial operations are possible using the Voronoi diagram. However, for some of these, the discrete Voronoi diagram has to be extended with additional functionality. In the next section the possibilities and necessary additions will be presented.

## 5.2. FUTURE POSSIBILITIES OF THE CREATEDISCRETEVD ALGORITHM

This research project has a limited time frame. Within the available time the concepts and functionalities that were discussed above have been implemented such as resampling functionality, natural neighbour interpolation and dynamic manipulation of seeds. However, there are several ideas, possibilities and concepts that were thought of during this project which can definitely aid in making the discrete 3D Voronoi diagram a useful data structure, especially in combination with off-the-shelf GIS. The most important of these future possibilities will be discussed in the next sub sections.

### 5.2.1. ADJACENCY AND EDGES

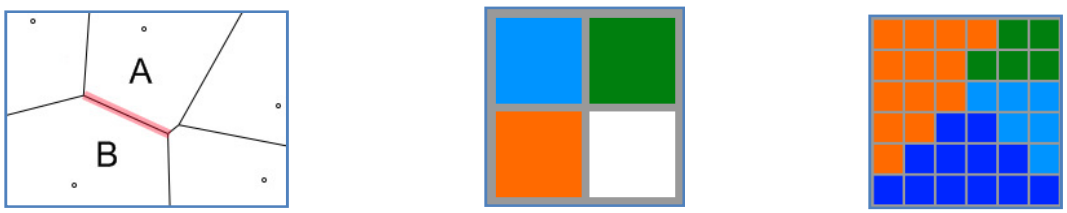
One of the great advantages of using exact objects is that the concept of topology can be added to those objects. In a general sense, these objects can constitute things that one could see as separate items. For instance: a house can be an object, a garage, a tree etc. In the specific case of the Voronoi diagram, an object would be a Voronoi cell.

Although the term topology can also be used in combination with continuous fields represented in raster structures, the two have different meaning. Topology in exact space describes "*the continuity of space and spatial properties, such as connectivity, that are unaffected by continuous distortion.*" (Burrough and McDonnel, 2006). One of these properties, aside from connectivity, is adjacency. In 2D, adjacent objects are objects that share a common boundary or edge, whereas in 3D objects are considered to be adjacent when they share a face.

In some data formats, the topology relationships between objects are stored within the file (Arc/INFO coverage files), in other formats (ESRI Shapefile) only the objects are stored, and topological relationships have to be calculated from the geometries in real-time (ESRI, 1998).

In raster representation, topology between objects can be defined as well, although the definitions are slightly different (Winter and Frank, 2000). However, the concept of adjacency is rather similar in both environments. In Figure 5.5a two polygons are adjacent: they share a common edge (edge connected). If two polygons would meet only at one vertex, they are also adjacent, but they are vertex connected.

The boundary between pixels is non-existent. They are drawn in most figures to emphasise that a pixel is a discrete unit, but in reality the boundary is not there. So how do we define raster adjacency? In Figure 5.5b four pixels are adjacent. These can also be edge connected (blue and orange pixel) or point-connected (orange and green pixel). But not only pixels can be adjacent. In the case of the discrete Voronoi diagram the Voronoi cells might also be adjacent. A Voronoi cell A is said to be adjacent to another Voronoi cell B, when at least one of the pixels in A is adjacent to at least one of the pixels in B. This concept is illustrated in Figure 5.5c, where the light blue, orange and dark blue Voronoi cells are adjacent to each other, but the green is only adjacent to the orange and light blue pixels. Thus one can say that a Voronoi cell is adjacent to another Voronoi cell, if at least one of each cell's pixels are adjacent (edge and point adjacency).




**(a)** **(b)** **(c)**

**Figure** 5.5. Adjacency in exact and discrete environments. **(a)** Polygon A and B are edge-connected. **(b)** The blue and orange pixels are also edge-connected, while the green and orange pixel are corner-connected. **(c)** Light-blue, dark-blue and orange areas are adjacent. Green area is only adjacent to orange and light-blue area.

The concept of adjacency can be integrated with the `CreateDiscreteVD` algorithm as follows. During the construction of the discrete Voronoi diagram, areas are dilated. During this dilation operation, an outer pixel *p* tries to push its ID value to its neighbour(s) *q*. If *q* has already been claimed, a check is performed to determine which value is the correct choice. If the result of this check is that the original value of *q* is correct, this means that that pixel is part of the boundary of the neighbouring area, and *p* is part of the boundary of the current area. By storing this information a database can be created that for each Voronoi cell stores both the boundaries of that cell, as well as the ID value of the adjacent Voronoi cells. This database can then be used to speed up existing operations such as the RemovePoint operation (see Section 4.5). It also clears the road for other adaptations and extensions to the algorithm.

### Extraction of Delaunay triangulation / tetrahedralization

One of the possibilities that the adjacency table could introduce is creating or extracting the Delaunay triangulation / tetrahedralization from the Voronoi cell adjacency table. By drawing lines between seeds that are adjacent (where adjacency is based on Voronoi cell adjacency (edge variant)), one has in principle the DT of a point set. Drawing lines and filling areas in a raster are quite straightforward operations (Gourret and Paille, 1987). After filling the triangles (2D) or tetrahedra (3D), the discrete DT is complete (Figure 5.6a and c). The discrete DT can also be used as a method to represent continuous field data for instance (Fuchida, Kashima et al., 2005)

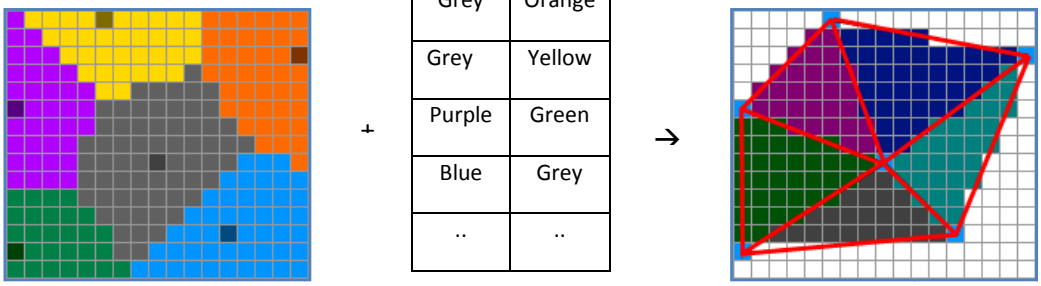



**Figure** 5.6. Creating Delaunay triangulation through adjacency table. **(a)** The Voronoi diagram. **(b)** The corresponding adjacency table. **(c)** The resulting Delaunay triangulation (red lines), and the corresponding discrete version.

### Faster `RemovePoint` algorithm

Another major benefit from this minor change of explicitly storing cell boundaries, is the possible increase in efficiency in the operation `RemovePoint` and `MovePoint`. The `RemovePoint` operation is as of yet not as efficient as it could be. The main issue is the lack of efficiency in finding the boundary Currently, the algorithm uses an internal function that returns the index of the pixels. It does this much more efficiently than a custom built function that evaluates the entire grid, but the mechanism is still the same. It then visits these pixels, recording the pixels that form the boundary of adjacent Voronoi cells. But if the database containing adjacency and edge information described above is stored with the 3D Voronoi diagram, the `RemovePoint` algorithm can be simplified since no boundary pixels have to be identified.

The general dilation function will simply then re-dilate from the list of boundary pixels, returning a Voronoi diagram that does not contain the deleted seed anymore. Figure 5.7a-c shows the method as it is now, Figure 5.7a and b shows the simplified method.

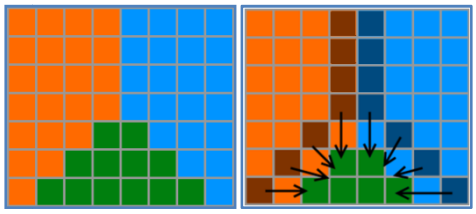



**Figure** 5.7. Removing a seed (and Voronoi cell) through adjacency table. **(a)** The Voronoi diagram. **(b)** The boundary pixels are already known, and can readily be re-dilated.

### 5.2.2. GENERALIZED VORONOI DIAGRAMS

The Voronoi diagram discussed in this thesis is the ordinary Voronoi diagram, which assigns areas (2D) or volumes (3D) to points based on Euclidean distance. Every location within this area is closest to the corresponding point than to any other. However, there are many more types of Voronoi diagrams that are also used quite extensively in different fields of science. Okabe, Boots et al. (1994) show how, with the help of 12 different generalized VDs, 35 different neighbourhood operations can be performed. The Voronoi diagrams they propose are generalized in space, assignment function (determines to which seed is a location assigned), and in the set of seeds that are used. Examples of these are the weighted Voronoi diagram and the high order (ordered) Voronoi diagram (Okabe, Boots et al., 1994). These Voronoi diagrams can be difficult to construct or modify in vector format in two dimensions (Gahegan and Lee, 2000; Dong, 2008), let alone in 3D. It is interesting to see if the discrete versions of these generalized Voronoi diagrams can also be created with the `CreateDiscreteVD` algorithm. To do this, some examples of a generalized Voronoi diagram will be presented and the suitability of the algorithm created in this project will be discussed.

#### Weighted Voronoi diagram

The weighted Voronoi diagram is a diagram in which the area that corresponds to a seed is defined by a weighted distance function. The weighted Voronoi diagram is an interesting diagram that can be used for instance to define the areas in which a certain product is ordered at the lowest price at supplier $p_i$, based on :

- the price of the product: $P_i$,
- the distance between supplier $p_i$ and a location $p$: $d(p_i, p)$
- the cost of transport $t_i$ per distance unit.

The total cost for ordering the product at location $p$, from supplier $p_i$ would then be $P_i + t_i d(p_i, p)$. The weighted Voronoi diagram would in this case show the areas in which the cheapest supplier is the supplier corresponding to that area (Okabe, Boots et al., 1994).

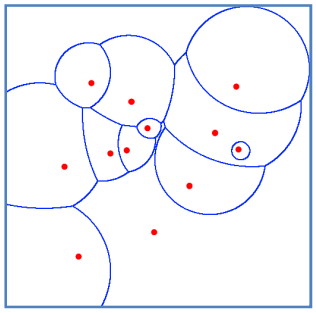
This weighted distance between a location $p$ and a seed $p_i$ can be described as follows:

$$d_w(p, p_i) = \frac{1}{w_{i1}}\, d(p, p_i) - w_{i2} \qquad \text{Eq. 7}$$

(From Okabe, Boots et al. (1994))

Where $d_w$ is the weighted distance, $w_{i1}$ would represent $\frac{1}{t_i}$ in the above example, and $w_{i2}$ would represent $-P_i$. Using only $w_{i1}$ would yield a so-called multiplicatively weighted Voronoi diagram (Figure

5.8). Using only $w_{i2}$ would yield an additively weighted Voronoi diagram. Using both would yield a compoundly weighted Voronoi diagram.



**Figur**e 5.8. Example of a 2D exact multiplicatively weighted Voronoi Diagram. (Adapted from http://info.wlu.ca/~wwwgeog/images/gambin2.jpg)

This distance equation can be used to easily create a weighted Voronoi diagram with the `CreateDiscreteVD` algorithm. This is primarily based on the fact that a seed can propagate itself to every pixel in the raster, as long as it does so according to the distance function and according to the `ARBITRARY_RULE`. For an ordinary Voronoi diagram the distance function is simply:

$$d(p, p_i) = \sqrt{(p_x - p_{ix})^2 + (p_y - p_{iy})^2} \qquad \text{Eq. 8}$$

But by substituting $d_w$ for $d$, the created Voronoi diagram will be weighted, and where $d$ can be called a Euclidean distance, $d_w$ can be called the Weighted Euclidean distance (Dong, 2008). From this, it becomes clear that each seed can have a different weight, and thus each seed must have its own distance function. This distance function is continuous, it must comply with the three conditions that need to be met in order to define a metric space (Section 3.2.2), and it is only valid only within the corresponding Voronoi cell. This can be generalized by stating the following:

***Postulate 1:***

**`CreateDiscreteVD` can create any Voronoi diagram for which the distance function belonging to a seed $p_i$ is describes a metric space within the boundaries of the corresponding Voronoi cell.**

Voronoi diagram in a river

With this statement in mind, it is also possible create a 'Voronoi diagram in a river' (Okabe, Boots et al., 1994). A 'Voronoi diagram in a river' can model for instance the areas of influence of a factory with respect to its smog output. This smog will not disperse radially, because wind will blow it into one

direction. An example of this can be seen in Figure 5.9, which show a 'Voronoi diagram in a river' in vector format.

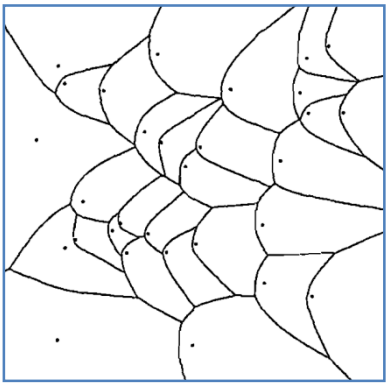


**Figure** 5.9. Example of a 'Voronoi diagram in a river'. It shows the non-radial dilation of a set of points, due to a directional influence. (From Okabe, Boots et al. (1994))

The distance function will be somewhat more intricate however. Let $v$ be the speed at which the smoke diffuse from the chimney, $w$ the windspeed in positive x direction and $\alpha = \frac{w}{v} < 1$. The distance function would then be:

$$d(p, p_i) = \frac{-\alpha(x-x_i) + \sqrt{(x-x_i)^2 + (1-\alpha^2)(y-y_i)^2}}{v(1-\alpha^2)} \qquad \text{Eq. 9}$$

(from Okabe, Boots et al., 1994) )

This distance function complies with postulate 1.

By combining the 3D 'Voronoi diagram in a river' and the weighted 3D Voronoi diagram, interesting applications in 3D can be devised. For instance; consider a river or estuary where a lot of factories are located along the shores which all discharge water with traces of heavy metals into the river. If one would be interested in the concentration of a certain heavy metal in the river, it is possible to model this using the Voronoi diagram in a river. The volumes of influence of each factory in the estuary or river can be calculated, taking into account the dispersion of the metals by the flow of the river. If some factories produce more discharge, this can be modelled by integrating a weight per factory. By using natural neighbour interpolation, one can model the 3D concentration of the heavy metals based on the discharge of the factories. Note: this is an illustrative example; to model such things as accurate as possible more variables have to be taken into account, such as flow mechanisms.

### Generalized Voronoi diagrams that cannot be created

There are also some generalized Voronoi diagrams that cannot be created using this algorithm. This is the case when the Voronoi diagram does not comply with Postulate 1. This the case with for instance directional network Voronoi diagrams, where the distance between two points is based on the available paths instead of the relative position of the two points. Therefore, it does not meet criterion 2 (Section 3.2.2, *metric space*) that states that the distance between 2 points should be equal when travelling in both directions, which is not necessarily so in a directional graph. From this it follows that a CreateDiscreteVD cannot create a Voronoi diagram with it. Other examples of Voronoi diagrams that cannot be created with this algorithm are k-order (ordered) Voronoi diagram and the Voronoi diagram with obstacles.

For in depth explanation of the generalized Voronoi diagrams, I refer the reader to Okabe, Boots et al (1994).

### Different metrics

Based on postulate 1, it should also be possible to create discrete Voronoi diagrams for different metrics, such as the Manhattan metric, of which the distance function is shown in Equation 3. This could be interesting for instance when one wants to find the nearest hospital in a city that has a gridded layout, or to perform any spatial analysis requiring a Manhattan Voronoi diagram for that matter.

### Voronoi diagram for objects other than points

The CreateDiscreteVD algorithm is created to work only with point datasets. However, it can be interesting to create the Voronoi diagram of other objects, such as lines, polygons or even polyhedra. Various people have already created algorithms for the exact Voronoi diagram of lines and areas in 2D (Drysdale and Lee, 1978; Kirkpatrick, 1979; Lee and Drysdale, 1981; Sharir, 1985; Yap, 1987; Dong, 2008). The 3D JFA algorithm by Rong and Tan (2007), is able to create a discrete 3D Voronoi diagram, however, it does this by creating several 2D Voronoi diagrams, and adding them together to form a 3D Voronoi diagram.

A Voronoi cell of an object other than a point shows the region in which every location is closer to the closest point of that object than to the closest point of any other object. An example of this is shown in Figure 5.10, where a 2D Voronoi diagram of areas is shown.

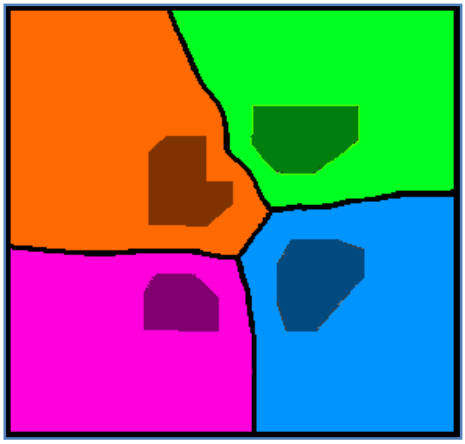


**Figure** 5.10. Example of a Voronoi diagram of areas.
(Adapted from Okabe, Boots et al, 1994)

The possibilities of the CreateDiscreteVD algorithm could be greatly increased if it can also create Voronoi diagrams for objects other than points; however, this is not the case. The problem lies in the fact that instead of the distance from a certain pixel to a point, the shortest distance from a pixel to an object must be determined. This is not possible with the current CreateDiscreteVD algorithm. However, other discrete Voronoi diagram algorithms have shown this potential in 2D, such as (Li, Chen et al., 1999; Fuchida, Kashima et al., 2005; Zhao, Li et al., 2008).

### 5.2.3. EXTENDED FILE STRUCTURE

To accommodate for the possible extension discussed above, especially concept of adjacency, not only the data itself has to be stored. Also, the adjacent seeds have to be stored, as well as the pixels that for the boundary of each Voronoi cell. It is quite straightforward that this will increase the required storage space quite drastically; however, the advantages that these features will provide are quite substantial. Therefore I would recommend extending the proposed data structure from Section 4.4.5 as follows:

- **A file containing the (3D) raster data**
- **A file containing the attribute values that correspond to the ID values of the seeds.**
- **A file containing the Ids of the adjacent seeds, for every seed:**
- **A file containing (for every seed), the indices of the pixels that form the boundary of the corresponding Voronoi cell.**

## 5.3. DISCRETE VERSUS EXACT

That the discrete 3D Voronoi diagram can be used to model continuous surfaces has been proven in the previous sections. However, the fact that it is possible does not mean that it is the best way. In the beginning of this research, the exact 3D Voronoi diagram was also introduced, and the analogues between the exact and discrete versions have been mentioned frequently, either to show the differences between them or their similarities. In this section, the discrete and exact version of the 3D Voronoi diagram will be compared as to how each method is equipped to model three dimensional continuous fields, and in general spatial analysis and operations. This will be based partly on their characteristics; some properties can make a method more suitable for a certain purpose than others. Also, concepts such as accuracy, speed and the ability to dynamically model the Voronoi diagram will be discussed.

### 5.3.1. SPEED

As stated in the scope of this research (Section 1.3.3), the aim of this research was not to create the most efficient algorithm possible. However, during the designing and the implementation of the `CreateDiscreteVD` algorithm, many tweaks have been made to make the `CreateDiscreteVD` algorithm as efficient as possible within the available time frame. By doing so, it was attempted to increase the suitability of this algorithm for creating 3D Voronoi diagrams. To see whether or not the `CreateDiscreteVD` is suitable algorithm to create the 3D Voronoi diagram, it is interesting to see how this algorithm performs in terms of speed, compared to an algorithm creating the exact version. The algorithm for the exact version is the same as is implemented in Ledoux (2006).

Although such a comparison seems relatively straightforward, in this case the comparison is not, since the end products of both methods are totally different objects. Perhaps a better comparison would be to see how the methods compare in speed, when the same end product is created by both methods, and both methods are created in the same language.

To get the same end result from both types of algorithm, one of them has to be modified. Since this research is focussed on the discrete 3D Voronoi diagram, this will be the desired output for both the discrete and the exact algorithm. The exact algorithm used to do this is an extension to the CGAL library (CGAL, 2009). As explained earlier, this method first creates the Delaunay tetrahedralization, and then derives the Voronoi diagram from that. To create a discrete Voronoi diagram with this method, first the Delaunay tetrahedralization is created. Then a function is called that gives the closest seed for any location. By doing this for every voxel in a 3D raster, a 3D discrete Voronoi diagram can be created. Conceptually however, this is not a very efficient way to do this.

Although this comparison seems to be based on more equal terms, it is still skewed. This has to do with two other factors. The first is the programming languages in which the exact and the discrete Voronoi diagram algorithm have been created in. Both the discrete and exact versions have been implemented in the Python language. However, the Python implementation of the exact 3D Voronoi diagram is based

on the CGAL library (CGAL, 2009) that has been written in C++, and for which so-called bindings are provided. These bindings allow Python to use the CGAL library, which allows for considerably faster execution of very intricate geometric calculations than a pure Python implementation would. The `CreateDiscreteVD` algorithm has been written and implemented almost completely with Python code and can thus be expected to run much slower.

### Cython

One technical aspect to increase efficiency is to write certain parts of the algorithm in Cython (Behnel and Bradshaw, 2008). By writing in Cython, parts of the program can be written in C instead of Python. For simple repetitive operations, this can increase efficiency by several orders of magnitude. Parts of the `CreateDiscreteVD` algorithm have been implemented in Cython, increasing the speed with a factor 20. If the algorithm itself could be written in C entirely, the efficiency would increase dramatically, especially when the array in which the raw data for the Voronoi diagram is stored can be converted to a C array.

In Figure 5.11, the running time for the `CreateDiscreteVD` algorithm is shown. From this graph it is clear that the time complexity of this algorithm is linear. An exact theoretical derivation of the time complexity for the `CreateDiscreteVD` algorithm is beyond the scope of this research.

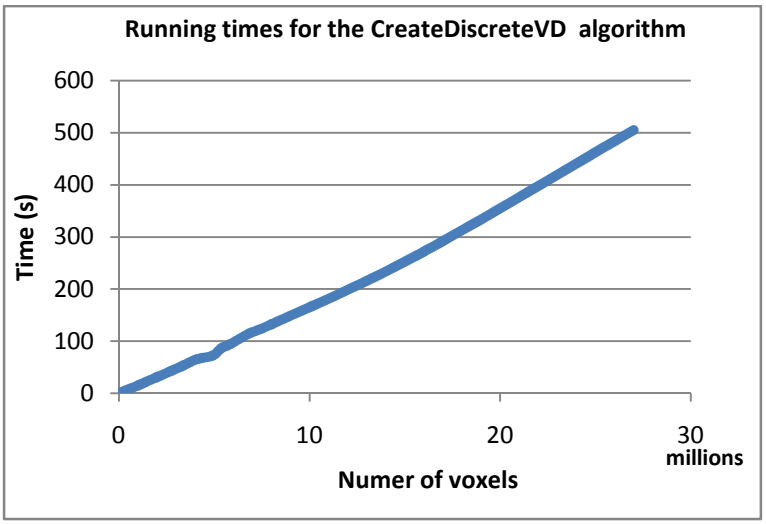



**Figure** 5.11. Running times for the `CreateDiscreteVD` algorithm.

The second factor for the incorrect comparison is the way the Voronoi diagrams are created. The time it takes for the `CreateDiscreteVD` to create a finished discrete Voronoi diagram is mostly dependent on the number of voxels in the image. The more voxels that have to be evaluated, the longer it takes. In the exact-to-discrete method, this is different. The first part, creating the Delaunay tetrahedralization, is dependent on the number of seeds; the second part is mainly dependent on the number of voxels in the

desired discrete Voronoi diagram. Therefore, it is difficult to compare the two methods. In theory, one might be faster when using very few seeds in a large raster; the other might be faster when a lot of seeds are used in a small raster.

The fact that this comparison is difficult to make quantitatively, does not mean that it cannot be made qualitatively. Actually this can be easily done. With a few test cases, it can already be stated that the `CreateDiscreteVD` is slower for almost every set of parameters, compared to the exact-to-discrete method. Only when the raster is relatively small (< 100x100x100 voxels) is implementation of the `CreateDiscreteVD` quicker.

However, this does not mean that the development of the `CreateDiscreteVD` is a futile undertaking, since the functionality of the algorithm adheres to the criteria set in the start of this thesis. It is expected that if the entire algorithm is written in C or C++, that it will be several orders of magnitude faster than it is now. That would make the algorithm more suitable for the creation of the discrete 3D Voronoi diagram, and perhaps it will be quicker than the exact-to-discrete method. The reason that an implementation in C or C++ will be quicker is that those languages are compiled, whereas Python is an interpreted language. In an interpreted language, a line of code is parsed, and then executed. Then the next line of code is parsed and executed. Compiled languages first parse the entire file into machine code. Once that is finished the machine code can be executed in one go, giving a significant speed improvement.

### 5.3.2. ACCURACY AND PRECISION

When comparing vector and raster representations, concepts that are frequently used are accuracy and precision. These terms are related, but they are not the same. Accuracy simply means how accurate a measurement or estimation (for instance in an interpolated image) is with respect to its true value. Whereas precision relates to how precise a measurement can be made. Can temperature measurements in the ocean be made in degrees, or tenths of degrees using a specific thermometer? This is usually controlled by the measuring equipment and the format in which data is stored. A measurement can be extremely precise, but is not necessarily very accurate.

#### Using different structuring elements

In Chapter 3 it is explained that in the process of dilation, a structuring element (or neighbourhood) is used to dilate a pixel. This structuring element can take any shape, but for this project, three structuring elements were chosen; the Moore neighbourhood (8-neighbourhood) and Neumann neighbourhood (4-neighbourhood) (Section 3.3). In several trials, the three neighbourhoods were compared in speed and accuracy. Although there was no difference in accuracy, with the Neumann neighbourhood the algorithm required the least time to complete. For this reason, in future test, only the Neumann neighbourhood was used. The graph Figure 5.12 shows the different calculation times for the three

structuring elements at different raster sizes. The tests were conducted using the 2D Voronoi diagram for practical reasons. However these results can be generalized for the third dimension as well.

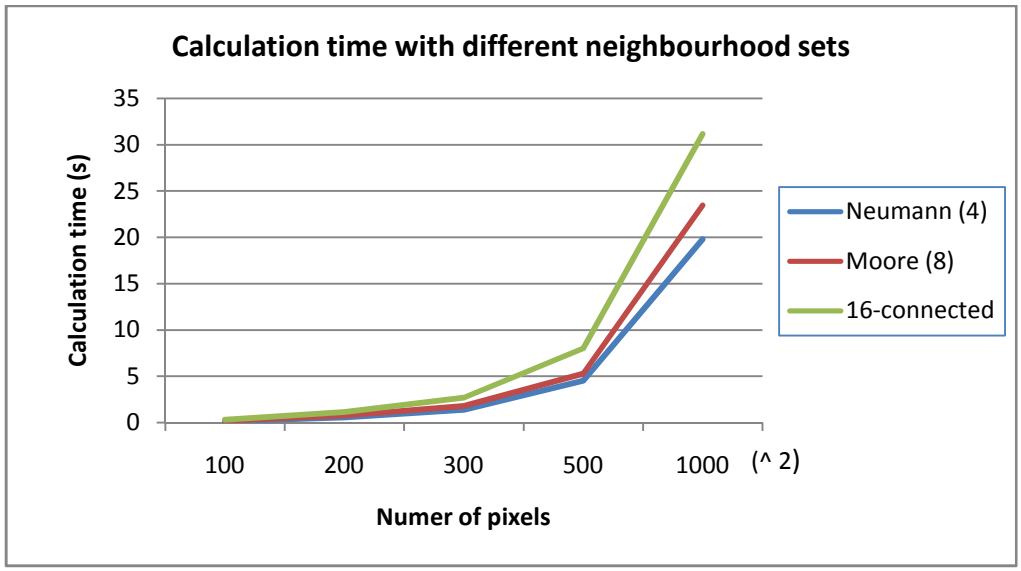



**Figure** 5.12. Calculation time of discrete 2D Voronoi diagram using different structuring elements

The reason that it takes more time to complete with larger structuring elements might sound paradoxical since more pixels can be claimed in one round. However, this also means that when two or more Voronoi cells start to converge, overlap between the dilated cells becomes more frequent, and this means that there are more pixels that need to be corrected, which also takes time. From these results, it is clear that this algorithm performs the most efficient with a structuring element that is as small as possible while still expanding in all directions.

## Shape representation

In comparing raster with vector representation, the first thing that is to be acknowledge is that a raster is always less accurate in representing the shape of an object than the vector representation is. For instance, consider a triangle (Figure 5.13a):

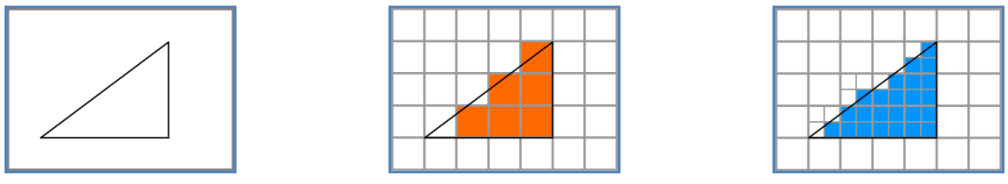



**Figure** 5.13. Comparing accuracy between exact and discrete objects. **(a)** An exact triangle. **(b)** Same triangle, now approximated by discrete representation. **(c)** Same as in **(b)**, but now with doubled resolution, yielding better approximation.

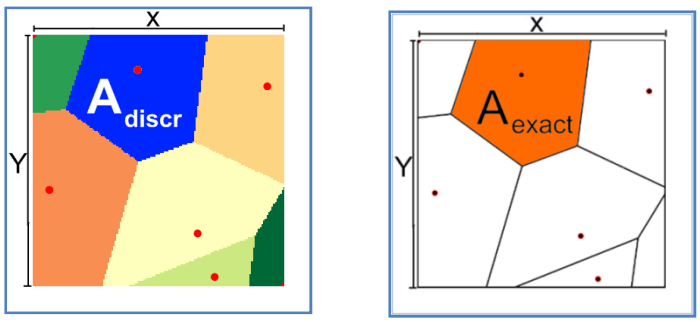
Figure 5.13a shows the triangle in vector format, Figure 5.13b and c in raster format. It is clear that the shape of the triangle is approximated in Figure 5.13b and c, but not really well. Based on Figure 5.13b, one could say that the triangle is an isosceles triangle, while in vector representation it is not. This approximation of a shape in raster representation can be improved by increasing the number of pixels. This decreases the spatial resolution; one pixel will represent less space in the reality it is modelling, but the shape of the object will approximate the exact shape more closely. This can be done indefinitely (within bounds of realistic computing practice) but the shape will simply never be exactly the same as the shape in the vector representation. This exposes one of the disadvantaged in raster data models: a higher degree of accuracy means more storage space is needed.

### Volume and area representation

An interesting feature of the discrete Voronoi diagram is the ease with which the area or volume of influence (Voronoi cell) of a point can be calculated. The only thing that has to be done is to calculate the number of pixels or voxels that make up the cell, and multiply this by the area or volume of one pixel or voxel respectively.

Theoretically, the accuracy problems that are illustrated in Figure 5.13b should have their consequences on the accuracy when performing these volume calculations. For testing purposes, two 3D Voronoi diagrams were created, one exact and one discrete Voronoi diagram. Both were created using the same dataset, with 17 randomly distributed points. A bounded Voronoi polyhedron was selected in the exact model. The ratio between the volume of this polyhedron and the bounding cube of the exact Voronoi diagram was created in was calculated ($r_{exact}$). This ratio is a constant, in other words, a bigger cube will yield exactly the same ratio, since it is created in a vector environment. In the discrete Voronoi diagram, the same polyhedron was selected. Also for the discrete version, the ratio between the volume of the polyhedral and the volume of the cube it was created in was calculated. This is illustrated in Figure 5.14, where for the sake of simplicity a 2D example is shown. In Figure 5.14, The total area of the image $A_{total}$ = X * Y. In the discrete version, $A_{discr}$ is the area of the blue Voronoi cell. In the exact image, $A_{exact}$ denotes the area of the corresponding exact Voronoi polygon (orange). To compare the accuracy of the discrete Voronoi diagram to that of the exact Voronoi diagram, one can define the two ratios $r_{discrete} = A_{discr} / A_{total}$ and $r_{exact} = A_{exact} / A_{total}$.

$r_{exact}$ will never change when increasing the size of the exact Voronoi diagram, but $r_{discrete}$ does change slightly when changing the size of the discrete Voronoi diagram. Increasing the number of pixels should increase the accuracy, and therefore the $r_{discr}$ should approach $r_{exact}$.




**(a)** **(b)**

**Figure** 5.14. Comparing accuracy between exact and discrete objects through comparing ratios **(a)** $A_{disc}$ is defined by the area of the blue polygon **(b)** $A_{exact}$ is defined by the area of the orange polygon

By comparing the $r_{exact}$ with $r_{discrete}$, the error in the volume of the discrete 3D Voronoi diagram can be calculated. In this case it is expressed as the amount the ratios differ in percentages. As mentioned earlier, to increase the accuracy, theoretically one can use more pixels. Therefore, the influence of increasing the number of pixels in the discrete Voronoi diagram has also been modelled. The outcome is shown in Table 5.1 and visualized in Figure 5.15.

| (Nr of voxels) ^ 3 | Difference (%) |
|---|---|
| 20 | 16.20 |
| 50 | 7.84 |
| 75 | 2.22 |
| 100 | 0.86 |
| 125 | 0.42 |
| 150 | 1.69 |
| 160 | 0.74 |
| 170 | 0.47 |
| 175 | 1.76 |
| 180 | 1.08 |
| 190 | 0.25 |
| 200 | 0.06 |
| 250 | 0.12 |
| 300 | 0.54 |

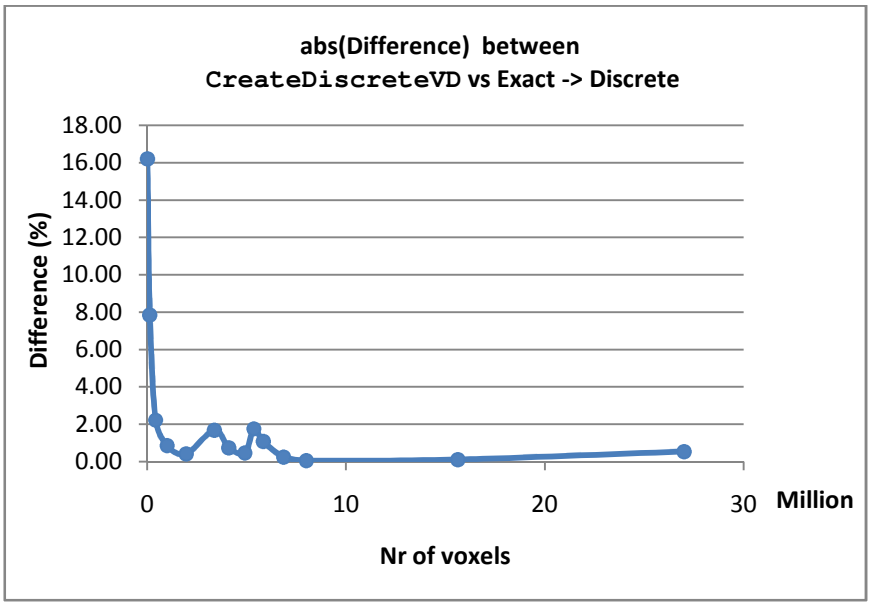


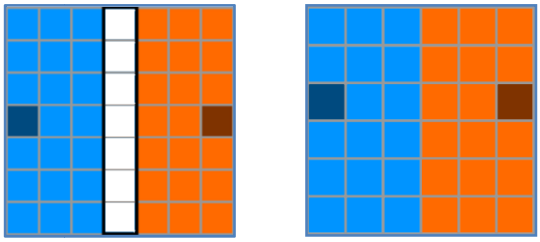

**Figure** 5.15. Graph showing the relation between error in volume calculation of a random chosen Voronoi Cell for a 3D discrete Voronoi diagram. The error is the absolute relative difference between the exact volume and discrete volume of the same Voronoi cell in percentages.

**Table** 5.1 The absolute relative error shown in Table form, with respect to the number of voxels used to create the discrete Voronoi diagram.

Table 5.1 shows a highly erratic outcome in the error column. Some errors had a negative value, but in the table the absolute value is shown. The peaks are caused by the conversion to absolute values. Would the graph show signed values, the resulting graph would zigzag around the y=0 line. In the following section I propose an explanation for the trend in the errors, and why the absolute value has been used as indicator.

There seems to be no correlation between the number of pixels used and the size of the error. The expected drop with increasing number of pixel is visible, when going from 20^3 voxels to 125^3 voxels. However, the drop does not continue and the error remains to fluctuate erratically between approximately -1% and +1%.

One reason for the fluctuation of the error is the fact that pixels that are equidistant to two or more seeds are assigned to the seed (and Voronoi cell) with the lowest ID value. When creating the discrete Voronoi diagram in different scales, as has been done with the Voronoi diagram mentioned in Table 5.1, a pixel that is equidistant to two or more seeds in a 50x50x50 raster might not be so in a 75x75x75 raster. Since the volume of only one Voronoi cell is calculated, in one resolution this cell might have slightly less pixels (relatively) than in a different resolution. This is illustrated in fig. Figure 5.16.



**(a)** **(b)**

**Figure** 5.16. Assigning equidistant pixels to a seed. **(a)** The centre column of pixels is equidistant. **(b)** Based on ARBITRARY_RULE, it is assigned to the blue seed.

In Figure 5.16a the centre column of pixels (with black outline) is entirely equidistant to both seeds (darker pixels). In this case, blue has an ID value 1 while orange has ID value 2. Based on the ARBITRARY_RULE, the centre column will be assigned entirely to the blue seed. The area of the blue seed is now $\frac{4}{7}$ th of the entire area, while in the exact version it would be exactly $\frac{1}{2}$. Since this error can be either negative or positive depending on the object and the chosen resolution, the absolute values was given in Table 5.1 to illustrate the magnitude of the error. When changing the resolution by 1 pixel in both x and y direction, as is done in Figure 5.16b, the ratio of the areas comes closer to the ratio in the exact representation (in this case they are exactly similar).

### Equidistant pixels

The error in volume or area explained above is a structural error. One way to decrease this structural error is to change the ARBITRARY_RULE, discussed in Section 4.3.1, which states what has to be done

when a seed is equidistant to two seeds. Normally, it is assigned to the seed with the lowest ID. But if the assignment would be random, then the probability that the pixel is assigned to either seed is 0.5 for both seeds, instead of 1.0 for the seed with the lowest ID. However, this is not possible as of yet with this algorithm. To understand this, it is important to realize that the rule of assigning the pixel to the seed with the lowest ID is rather arbitrary: it could also be the seed with the highest ID or any other rule. However, it has to be a rule that defines a suitable seed in a reproducible way: a specific situation must have the same outcome every time, otherwise the algorithm will fail. A situation where pixels are randomly assigned does not fit this criterion; therefore it is not suitable for now.

The reason why the algorithm needs a reproducible assignment rule, and cannot assign a seed randomly is illustrated by Figure 5.17, in which equidistant pixels are arbitrarily assigned to a seed.

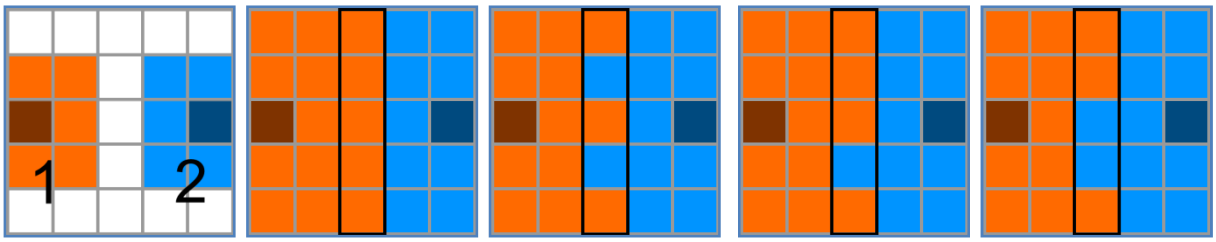



**Figure** 5.17. Random assignment of equidistant seeds. **(a)** Dilation of two seeds. **(b)** Centre column is assigned to orange seed, since they contained NoData, and the orange seed came first. **(c)** When the blue seed is dilated, some pixels in the centre column are randomly assigned to the blue seed. **(d)** When re-checking neighbours, the orange seed reclaims some pixels randomly. **(e)** Same thing happens for the blue seed, entering a never ending loop.

In Figure 5.17a, two seeds have been dilated once using the Moore neighbourhood. The column of pixels in the centre of the image contains only equidistant pixels. In the following round (Figure 5.17b), the orange pixels start dilating, and claims all the pixels in the centre column, because they contain NoData. In the same round, after the orange dilation, the blue area dilated. When it evaluates the equidistant pixels, it randomly assigns some pixels to the blue area (Figure 5.17c). In the next round the neighbours of the changed orange pixels are evaluated, including the pixels assigned to the blue area. It might be that these are than assigned to the orange area again (Figure 5.17d). This process can go on indefinitely (Figure 5.17e), which means the algorithm is in an infinite loop. This is the reason why the random assignment has not been implemented

### Accuracy in natural neighbour interpolation

Because the natural neighbour natural neighbour interpolation method is directly connected to the area (in 2D) or volume (in 3D) of the different Voronoi cells, any error that these Voronoi cells contain will be reflected automatically in the natural neighbour interpolation. An example of this can be seen in Figure 5.18, where a Voronoi cell of point q has an area that is 7% smaller for the discrete version compared to

the exact version. This will directly reflect in the value that is assigned to point q after natural neighbour interpolation.

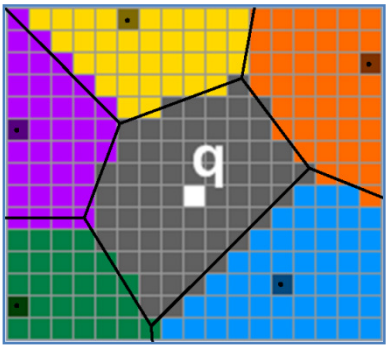



**Figure** 5.18. A portion of an Exact and discrete Voronoi diagram in overlay. The area of the discrete Voronoi cell of point q is 7% smaller than the exact Voronoi cell of point q.

### Resolution of the discrete 3D Voronoi diagram

When looking at the relation between resolution and shape accuracy, it is clear that increasing the resolution can increase the accuracy. But what can be said on the required resolution of the discrete 3D Voronoi diagram, in order to get a satisfactory result?

Obviously, this depends strongly on the purpose for which the Voronoi diagram is built. One could state for instance that at least every sample point must be represented by a single pixel, so that all sample points are represented in the diagram. This concept can be expressed with the help of the Nyquist sampling frequency. The Nyquist sampling frequency is the highest occurring frequency in the dataset times two (Baxes, 1994; Hengl, 2006). When applying this relation to a set of geographically referenced points, the relation is as follows:

$$p \leq \frac{\bar{h}}{2} \qquad \text{Eq. 10}$$

Where $p$ is the spatial resolution, and $h$ the average distance between two smallest point pairs. However, this requires one to know the spatial distribution of the sample points. To retrieve this information, first the entire dataset has to be examined, requiring a substantial amount of time. Hengl (2006) postulates some more relations for the spatial resolution, and there is one that is perhaps the most suitable for the discrete 3D Voronoi diagram, which gives the relation between the minimum spatial resolution, the area and the number of samples collected for a random sampling distribution in 2D. This relation can be generalized to three dimension, which will yield the following equation

$$p = C \sqrt[3]{\frac{V}{N}} \qquad \text{Eq. 11}$$

Where p is the spatial resolution in meters, C is a constant, V the volume of the sampled area, and N the number of samples. For a random sampling, C must have a value of at most 0.25 (Hengl, 2006). Although sampling in geosciences in by all means not generally based on random sampling, it will resemble an anisotropic sampling distribution better than a regular sampling.

The relation in Equation 11 will ensure that every voxel in the discrete 3D Voronoi diagram will represent only one unique sample point. In other words, not seed (or sample point) will fall in same voxel with another sample point.

However, one can also argue that samples that are relatively close might be rather similar and can be aggregated (this obviously depends very much on the manner in which the area was sampled. With progressive sampling for instance the opposite is true). Or one simply wants to quickly create a coarse Voronoi diagram, to have a fast overview. Whatever the reason for it, it is already accommodated for in the `CreateDiscreteVD` algorithm. Sample points that occupy the same pixel are aggregated into a so-called compound point (Section 4.4.4). Such a point has the average coordinates of the sample points that are aggregated, as well as an averaged value. This allows for low resolution discrete 3D Voronoi diagrams, without having to completely ignore sample data.

#### Physical limitations

In the current setting, the algorithm cannot handle rasters that are bigger than 300x300x300 voxels. This is because of the memory limitations that the Numpy module has.

### 5.3.3. INTERACTIVE OR DYNAMIC MODELLING

GIS are nowadays not used only for importing data and performing spatial analysis on it. There is a growing trend that users need a more dynamic approach to modelling where the user is able to manipulate, modify or transform the data (Bailey and Gatrell, 1995; Gahegan and Lee, 2000; Ledoux, 2006; Petit, Claramunt et al., 2008). Through interactive, or dynamic, modelling, data can be probed, manipulated and assessed.

The prototype GIS created by Ledoux (2006) accommodates for the dynamic modelling of points. In other words, after the data was loaded and the exact Voronoi diagram was created, the points in the data structure can be moved, removed, or new points could be added. The data structure is updated locally instead of the entire structure having to be recreated. The main advantage of this is that the size of the dataset is not relevant for the operation in terms of efficiency or speed, since only points that are nearest neighbour to the modified point are used.

To accommodate for the same type of dynamic modelling, the `CreateDiscreteVD` was extended with the AddSeed, RemoveSeed and MoveSeed algorithms. These can only be used within the Python program, but if the `CreateDiscreteVD` algorithm would be integrated into GRASS using a plug-in for instance, this could make the modelling even more dynamic.

### 5.3.4. General Usability of exact and discrete 3D Voronoi diagram

The contest between vector and raster representation is one that is not going to end soon, since each data model has its own pros and cons. This is why there is no decision in this thesis on which is better: the exact or the discrete 3D Voronoi diagram. They both have their pros and cons, and are therefore suited to cope with different situations.

Since both representations represent the same geometric structure, both the exact and discrete 3D Voronoi diagram have some similar advantages and disadvantages. However, it is more interesting to see what the specific pros and cons are for both the data structures. Therefore, the pros and cons for both the discrete and exact version of the 3D Voronoi diagram are listed in Table 5.2. From it, it becomes clear that both implementations of the Voronoi diagram have their own strengths and weaknesses. Calculation time is also mentioned in the table, but the reader must keep in mind that the efficiency of the algorithm can be greatly increased by programming it in a different (lower-level) language.

| **Exact** | | **Discrete** | |
|---|---|---|---|
| Pros | Cons | Pros | Cons |
| - Enables high precision and accuracy<br>- Topology relatively easily added<br>- Easy to make spatial queries if topological data structure is used<br>- Once calculated, it is easily scalable (i.e. resizable)<br>- Rotationally invariant<br>- Relatively fast to implement (compared to discrete algorithm) | - Very difficult to implement robustly<br>- Difficult to apply dynamic modelling (add/remove points)<br>- Can be difficult to visualize<br>- Performing robust natural neighbour interpolation is expensive<br>- Creating generalized Voronoi diagrams is expensive, and has not often been done in 3D. | - Raster format is widely used<br>- Visualization is relatively easy and clear<br>- Dynamic modelling easily implemented (add/remove points)<br>- Straightforward in generating Voronoi diagrams in different metrics or other generalized Voronoi diagrams<br>- Easily combined with other (2D or 3D) rasters.<br>- Map algebra can be applied<br>- Array-like structure allows for relatively simple calculations | - Less accurate (depending on resolution)<br>- Only very limited topology possible<br>- Storage space is relatively high<br>- Resizing the raster is relatively expensive<br>- Rotationally variant<br>- Current implementation is relatively slow |

**Table** 5.2 General pros and cons for both the discrete and abstract version of the 3D Voronoi diagram

When looking at the difference in Table 5.2, some interesting differences can be seen. The exact Voronoi diagram is quite suitable for the ordinary 3D Voronoi diagram, especially if one is interested in for instance adding topology to the Voronoi diagram. However, the discrete version is well equipped for the generalized Voronoi diagrams, as well as for dynamic modelling. One other major advantage is the map algebra which is possible to apply on the discrete 3D Voronoi diagram. Although storage space and memory usage are many times higher for the discrete Voronoi diagram, the technical innovations in these areas allow for such large data structures.

## 5.4. GIS AND THE DISCRETE VORONOI DIAGRAM

In the previous sections, most research questions have been addressed. Based on the results, it is possible to address the main research objective: "*To assess the use of the discrete 3D Voronoi diagram for the modelling of 3D continuous information in geosciences*". This chapter will cover this assessment, as well as other (future) possibilities the 3D discrete Voronoi diagram in general, and specifically the `CreateDiscreteVD` algorithm has to offer.

### 5.4.1. SHOWCASE: GEOLOGICAL DATASET

To show the possibilities of the discrete 3D Voronoi diagram, some examples of its usage will be shown here. The dataset that is used is a geological dataset, containing the concentration of a chemical substance. It exists of a set of 150 data points, comprised by x, y, z coordinates and an attribute value. Normally, a set of points would be much larger, ranging from 10.00 to 100.000 points. However, for this showcase, a small dataset is used to be able to show details clearly. In Figure 5.19 the distribution of the sample points in this test data set is shown. It can be seen that most of the points are taken along lines parallel to the z-axis (blue), which indicates that the data is taken from for instance drilled wells. The anisotropy of the distribution of the sample points is also clear from this image.

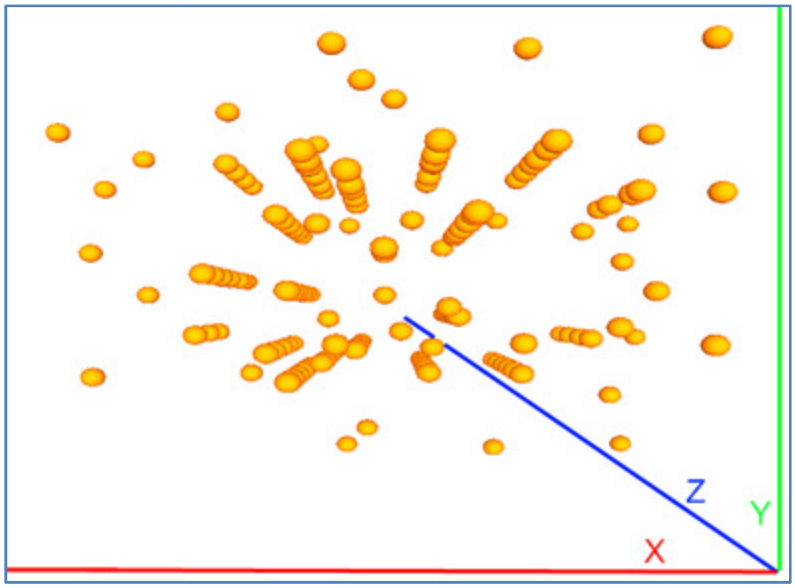



**Figure** 5.19. Distribution of the points in the test dataset. The anisotropy in distribution is clearly visible.

The first step is to calculate the discrete 3D Voronoi diagram using the `CreateDiscreteVD` algorithm. After this is done, it can be visualized in MayaVi2. In Figure 5.20, five different visualization methods are shown for specific attribute values; surface visualization, using a controllable cutting surface (slicing), two different visualizations using isosurfaces and a volume rendering.

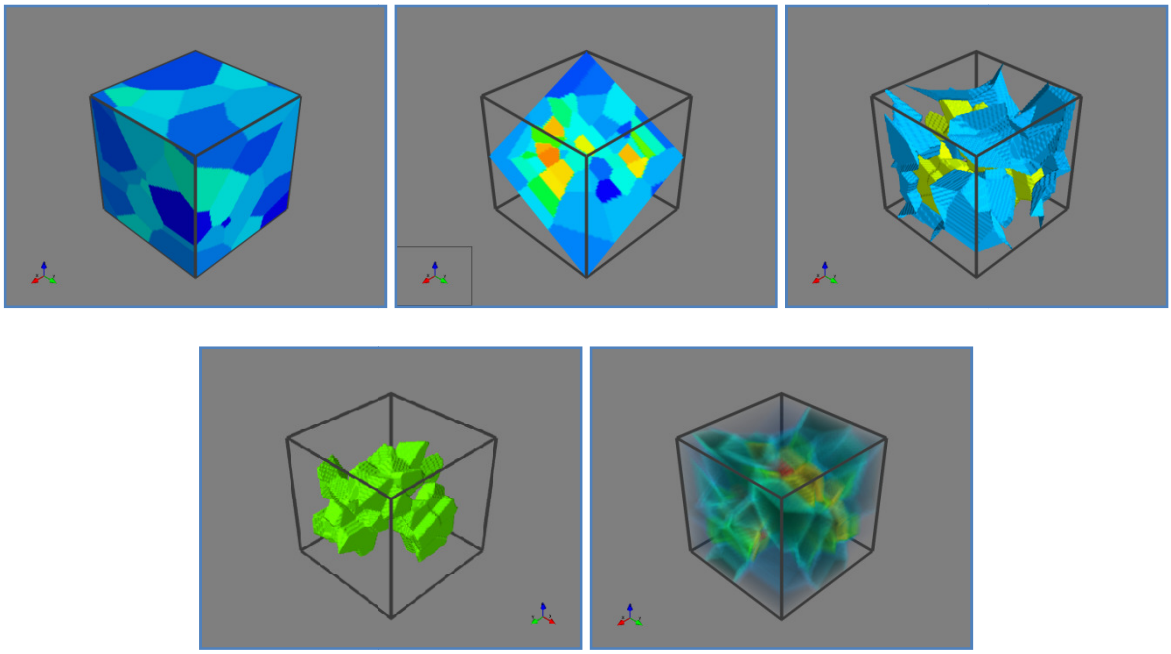


**Figure** 5.20. Visualizing the discrete 3D Voronoi diagram. **(a)** Surface visualization. **(b)** Slicing. **(c)** 2 different isosurfaces. **(d)** Another isosurface. **(e)** Volume rendering.

It is clearly visible that the different visualization methods have their advantages and disadvantages. The slicing method gives a good indication of the distribution of points and their neighbourhood relations, while the isosurfaces can identify the location of specific features or anomalies. It is also clear that volume rendering is not the best method of visualizing 3D raster, as already explained in Section 2.2.

After visualizing the discrete 3D Voronoi diagram, it might be interesting to see what the continuous field that the data describe looks like. This is done by interpolation the discrete 3D Voronoi diagram using the implemented natural neighbour interpolation method. The results are visualized in Figure 5.21. Note that the isosurfaces are created based on the same values as in Figure 5.20.

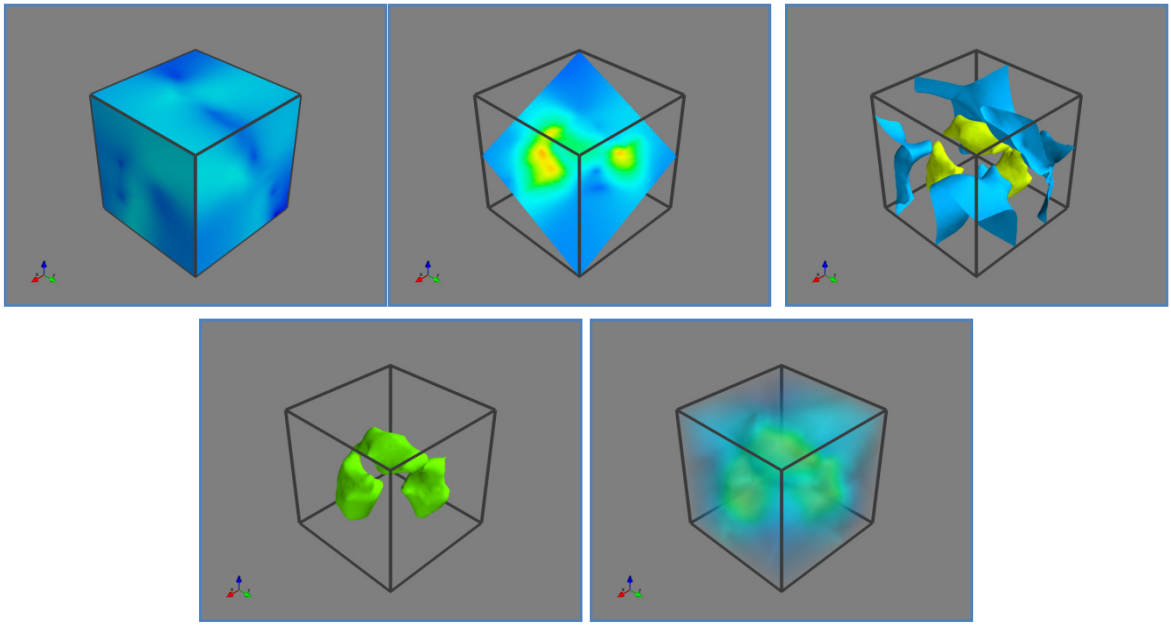


**Figure** 5.21. Visualizing the interpolated volume.**(a)** Surface visualization. **(b)** Slicing. **(c)** 2 different isosurfaces. **(d)** Another isosurface. **(e)** Volume rendering.

From most of the methods shown in Figure 5.20 and Figure 5.21, it is clear that the natural neighbour interpolation method yield very smooth isosurfaces, as is expected based on Section 4.6. A interesting difference between the Voronoi diagram and the interpolated volume is visible in the isosurfaces. In the Voronoi diagram it is visible that the isosurfaces tend to follow the edges of the boundary voxels of the Voronoi cells, whereas in the interpolated volume the isosurfaces are smooth. This is because the values between the boundary voxels of two Voronoi cells can be relatively far apart, resulting in very distinct isosurfaces. In the interpolated volume however, the difference in values is spread over a larger area, resulting in smoother isosurfaces.

With the results from the previous two steps, it is possible to analyze and modify the discrete Voronoi diagram and the interpolated 3D raster. The map algebra function (Tomlin, 1990) enables users to perform intricate calculations and operations on the 3D grid. It can, for instance, mask a raster with

another raster. This would be interesting if one would want to show only a particular section of the raster, based on the value of each voxel. It is also possible to create neighbourhood (local) filters. To create these filters, a function is provided that enables the user to incorporate values of neighbouring voxels for each evaluated voxel. Using this function, it is possible create for instance raster representing the derivative of a continuous field. An example of such in 2D would for instance be a map representing the slope of a DEM.

Reclassification is also one of the possibilities that the map algebra function provides. Suppose a user would want to divide the attribute values in to three classes; lowest, medium and highest values. This can be easily done using map algebra functionality. As an example, this is done in Figure 5.22, showing a cut plane through the reclassified 3D raster (based on the interpolated volume). The reclassification was done in GRASS, and the visualization in MayaVi2.

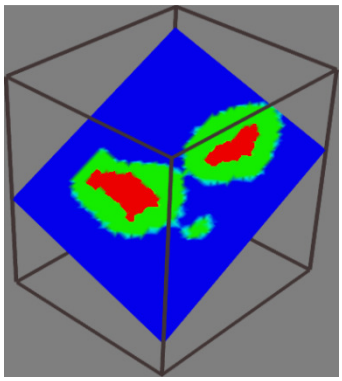


**Figure** 5.22. A cut plane through a reclassified 3D raster, showing a reclassification into 3 classes.

### 5.4.2. Evaluating the practical use of the discrete 3D Voronoi diagram in a GIS

The showcase presented above only shows a part of what the possibilities with 3D rasters, and especially with discrete 3D Voronoi diagrams are. The map algebra functionality, although not easy to master, gives a wide range of spatial analytical functionality. Now that these practical possibilities have established, a full description of the possibilities of the discrete 3D Voronoi diagram can be presented. To do this, the comparison between the discrete 3D Voronoi diagram and the exact counterpart will be used.

In the introduction a number of ways have been mentioned how the exact 3D Voronoi diagram can be used in GIS (Section 1.2.3). Among these are:

- To establish neighbourhood relations between points (especially useful for anisotropically distributed data).
- To determine the area of influence of points.
- To support numerous visualization and spatial analysis operations.
- As a prerequisite for natural neighbour interpolation

Of these three uses, the first three two capture a good deal of the operations that a user normally performs using a GIS. This is one of the reasons that Ledoux (2006) proposed to use the exact 3D Voronoi diagram as a data model for GIS, instead of simply using vector or raster data. The forth reason, as a prerequisite for natural neighbour interpolation also supports Ledoux's proposal; if the continuous field is required, it can be generated using natural neighbour interpolation.

To be able to assess the use of the discrete 3D Voronoi diagram in the use of modelling continuous fields using GIS, it is interesting to compare how the discrete 3D Voronoi diagram performs on the four methods shown above, and in other fields.

Establishing neighbourhood relations between seeds

There is not much difference in the way the exact and the discrete 3D Voronoi diagram handle these relations. Adjacency is in this case the key, which, for the exact version, can be determined quite easily from the Delaunay triangulation which is also created in the method Ledoux describes. In the discrete representation of the 3D Voronoi diagram, although not implemented yet, storing adjacency is easily added as a functionality, without having to perform intricate calculations.

Determining the area of influence of a point.

For the exact 3D Voronoi diagram, the Voronoi cell (which represents the area of influence) is described by polyhedra, where as in the discrete Voronoi diagram, it is determined by the collection of voxels that carry the same value. This means that it is relatively easy to determine the area of influence by simply finding all voxels that carry the same value. In this way, sub-selections can easily be made, and be used in combination with different spatial analysis operations, such as map algebra, reclassification or interpolation methods. Therefore, I believe that in most cases, the discrete 3D Voronoi diagram is better-suited than the exact Voronoi diagram, in terms of area of influence of points.

Supporting numerous visualization and spatial analysis operations.

The exact Voronoi diagram can be less clearly visualized, as explained in Section 2.2. As for the spatial analysis operations, it has already been mentioned that Voronoi cells are easily identified, and since the raster format is used, performing spatial analyses on these structures is relatively easy. This is one of the points that the strength of the discrete 3D Voronoi diagram may lie in, since these spatial analysis operations comprise of the majority of operations, such as map algebra, interpolations, reclassifications, and other local, and focal operations, performed by a general GIS scientist.

Also, the `CreateDiscreteVD` algorithm has great potential in creating several generalized Voronoi diagrams, which is something that can be difficult to implement in algorithms creating exact Voronoi diagrams.

As a prerequisite for natural neighbour interpolation.

It has been shown that the `CreateDiscreteVD` is well equipped for creating a natural neighbour interpolation using the natural neighbour method, since the algorithm allows for interactive, local modification of the Voronoi diagram. Since the objective of the research is to see how the discrete 3D Voronoi diagram handles continuous field data, the method that enables the creation of the field is rather important. So, in this area, the discrete 3D Voronoi diagram is also better equipped than the exact 3D Voronoi diagram, since there are issues concerning the implementation of the natural neighbour interpolation using the exact Voronoi diagram.

Accuracy

In Section 5.3..2 it has been shown that the accuracy of the discrete 3D Voronoi diagram is inherently worse that the accuracy of the exact 3D Voronoi diagram, in terms of shape, and volume calculations of a Voronoi cell. However, this has always been an issue in the ongoing discussion between raster vs. vector. However, approximating accurate values is possible, by creating a discrete Voronoi diagram using the correct resolution. A way has been presented two determine the number of voxels required for an acceptable 3D discrete raster. This means that although the exact Voronoi diagram is more accurate, the discrete Voronoi diagram is still usable when applied correctly.

## 5.5. SUMMARY

In this chapter, the suitability of the discrete 3D Voronoi diagram has been assessed both in light of modelling continuous data within the environment of a GIS, as well as its (future) possible uses in other areas. The discrete 3D Voronoi diagram has also been compared to the exact 3D Voronoi diagram in several fields, and a list of advantages and disadvantages has been compiled or each

The discrete 3D Voronoi diagram can provide a simple but effective tool for the modelling 3D continuous field data in combination with a GIS. The implemented algorithms provide the creation of the Voronoi diagram, and the natural neighbour interpolation of the Voronoi diagram, resulting in a 3D continuous raster. This can be used in combination with the GRASS GIS package, to perform for instance map algebra. Although this can be complex, it does give a powerful tool for the manipulation of 3D raster images. The visualization options in MayaVi2 deliver powerful visualization tools with a user-friendly interface, thereby allowing for easy and accurate visualization of 3D raster images.

Some possible extension to the algorithm have also been presented for future development, such as the possibility of extending the data structure of the discrete 3D Voronoi diagram with a list of Voronoi Cell edges, for the extraction of the Delaunay triangulation, and the quick deletion of points. Also, the possibility of using this algorithm for the creation of generalized Voronoi diagrams such as the weighted Voronoi diagram and the ‘Voronoi diagram in a river’ has been discussed. It seems that the `CreateDiscreteVD` is suitable for the creation of these generalized Voronoi diagrams in 3D.

The exact and discrete 3D Voronoi diagram have also been compared on different issues, such as speed, accuracy, the ability to interactively model data and the general usability of both in handling spatial data. As can be expected, both representations have their own pros and cons. Apart from speed issues, both seem relatively suitable for most applications. However, the downside of the exact Voronoi diagram is that it can be bothered by degenerate cases in most operations such as the creation of the Voronoi diagram itself and when deleting points. The implementation of the natural neighbour interpolation is also not straightforward with the exact 3D Voronoi diagram.

The discrete Voronoi diagram has no issues regarding degenerate cases. However, it currently does suffer problems in terms of calculation time, making it not a practical tool yet. But it is expected that if implemented in a low-level programming language like C or C++, the algorithm will perform much faster, and will likely be suitable for practical use.

A small showcase has also been presented in this chapter, to illustrate the possibilities that the discrete 3D Voronoi diagram has when used in combination with a GIS. It showed the possibilities in visualization and in analytical functionality such as map algebra.

At the end of the chapter, the full range of possibilities of 3D raster images and the discrete 3D Voronoi diagram especially are presented, also in comparison to the exact Voronoi diagram. This overview also establishes that the discrete 3D Voronoi diagram is a powerful tool in modelling 3D data in earth sciences.

# 6. CONCLUSION

The main objective of this research is:

> *To assess the use of the discrete 3D Voronoi diagram for the modelling of 3D continuous information in geosciences.*

The research done to achieve this objective has been described in this thesis, and based on the results of this report, a set conclusion can be formulated. These conclusions are presented following the general structure of the sub-questions and objectives presented in Section 1.3 – Research Objectives.

## 6.1. DISCRETE VS. EXACT

The characteristics of the discrete 3D Voronoi diagram and exact 3D Voronoi diagram have been compared by performing an extensive literature review. From this it has become clear that the exact 3D Voronoi diagram is a data structure that can be difficult to implement. A robust and efficient implementation of the exact Voronoi diagram can be inhibited when so-called degenerate cases occur; situations in which data points are located in such a manner that extra intricate calculations must be performed to create a Voronoi diagram that both valid and at the same time can be dynamically modified.

Besides the exact Voronoi diagram, several methods have been examined that are aimed towards the creation of a discrete Voronoi diagram. None of these methods are inhibited in any way by degenerate cases, which is a large advantage compared to the exact Voronoi diagram. This can be considered to be the main difference in terms of the implementation of both structures. In practice there are more differences, which will be discussed in 6.5.3.

## 6.2. DISCRETE VORONOI DIAGRAM METHODS

First of all, the literature review on the discrete Voronoi diagram showed that compared to the exact Voronoi diagram, the discrete Voronoi diagram has not been extensively researched. Also, of all methods that were discussed, only one method was specifically developed for the discrete 3D Voronoi diagram, the rest were aimed towards the 2D Voronoi diagram.

Two different types of method can be distinguished: implicit and explicit methods. Implicit methods assign a voxel to a seed based on the value of the pixel next to them. Explicit methods explicitly determine the closest seed for each voxel. Explicit methods are not error-prone, but need a mechanism that will increase efficiency in order to be practically implementable. Implicit methods are theoretically efficient since distance calculations are not required for every voxel. However, a control mechanism is needed to ensure that voxels are assigned correctly.

## 6.3. THE NEW CREATEDISCRETEVD ALGORITHM.

Based on the literature review, two different methods were deemed both efficient and conceptually simple; the nearest neighbour sweep circle algorithm (implicit method), and the method based on the nearest neighbour search through the use of a kd-tree. However, the latter method could not be implemented during the timeframe available for this thesis, due to the lack of certain programming libraries.

The implementation of the new `CreateDiscreteVD` algorithm is based on the morphological dilation principle and on Schueller's nearest neighbour sweep circle algorithm. It differs from Schueller's algorithm in the fact that that when assigning voxels to a seed, no checks are made to see which neighbours should or should not be claimed. Only when there are situations in which a voxel is claimed by multiple seeds will distance calculations be performed. This algorithm was first implemented in 2D, and was later altered so it could create the discrete 3D Voronoi diagram. It has been proven that this algorithm will yield a valid discrete Voronoi diagram.

The algorithm has also been extended so that it is possible to add, move and remove seeds. The dilation principle shows that it is very well suited for the local insertion and deletion of points without having to recreate the entire Voronoi diagram. Being able to add and remove seeds is allows for the dynamic modelling of the 3D (continuous) data, which is a frequently used in current day modelling practices.

## 6.4. GIS AND FUNCTIONALITY

In order to assess the use of the discrete 3D Voronoi diagram in modelling geo-data, the GRASS GIS was chosen as test environment. This selection was based on several criteria such as the ability to handle 3D raster data, analytical functionality and visualization options. GRASS is currently the only GIS that can handle 3D raster images, and that enables analyses and visualization of the entire raster.

Although GRASS does provide 3D visualization options, these are user-unfriendly and contain bugs. Therefore a scientific 3D data visualization program called MayaVi2, which is very user-friendly, has been used for visualization purposes.

Functionalities have been identified that were not available in GRASS, but which could provide extra flexibility or added value to the discrete 3D Voronoi diagram. These functionalities are: 3D raster resampling and 3D natural neighbour interpolation based on the discrete 3D Voronoi diagram. These two functionalities have been developed and implemented, along with routines that enable the import/export of 3D raster data between the different programs.

## 6.5. UTILIZING THE DISCRETE 3D VORONOI DIAGRAM

### 6.5.1. SPEED

The implementation of the `CreateDiscreteVD`, but especially the natural neighbour interpolation is as of yet to slow to be suitable for general implementation in ordinary GIS. This is due to the speed reduction that is a result of using an interpreted programming language instead of a compiled language. Therefore, when discussing suitability of these algorithms, speed is not considered to be the most important factor.

### 6.5.2. SHOWCASE

The implementation of the `CreateDiscreteVD` algorithm combined with the functionality of GRASS, allows for an assessment of the main research objective. The suitability of the discrete 3D Voronoi diagram in the modelling of 3D continuous information is presented through a showcase, in which a geological dataset of 150 data points is used to create a discrete 3D Voronoi diagram. The Voronoi diagram is also interpolated using the implemented natural neighbour interpolation algorithm. The discrete 3D Voronoi diagram itself is very well equipped to display the distribution of the data points, whereas the interpolated volume yields a 3D raster containing an approximated continuous field based on the data points provided. Both structures are easily imported into GRASS, in which the can be combined with other 3D or 3D rasters (and vector images), and where for instance map algebra can be applied. This allows for reclassification, filter creation and other intricate calculations.

### 6.5.3. SUITABILITY OF EXACT AND DISCRETE 3D VORONOI DIAGRAM

The suitability of the exact 3D Voronoi diagram and the discrete 3D Voronoi diagram have also been compared in several areas: adjacency, determining area of influence of a point, visualization, natural neighbour interpolation and accuracy. Although both the discrete and exact 3D Voronoi diagram have their pros and cons, I find that the discrete 3D Voronoi diagram is equally or better suited most areas.

### 6.5.4. FUTURE POSSIBILITIES

For future work on the discrete 3D Voronoi diagram in general, I would advise to implement this conceptually algorithm simple algorithm in a low-level compiled programming language. Another promising possibility that was not available for this thesis was the construction of the discrete 3D Voronoi diagram through the use of the kd-tree.

For the possibilities concerning specifically the `CreateDiscreteVD` algorithm, the following extensions are very simple to implement, but would increase the use of the algorithm greatly:

- Adding Voronoi cell adjacency
    - Extraction of the discrete 3D Delaunay triangulation.
    - Faster RemoveSeed algorithm
- Including support for generalized Voronoi diagrams such as the weighted Voronoi diagram and the ‘Voronoi diagram in a river’.

- Voronoi diagram for different metrics

## 6.6. GENERAL CONCLUSION

Therefore, based on the following facts:

1) It is possible to create a discrete 3D Voronoi diagram using the `CreateDiscreteVD` algorithm.
2) It is possible to interpolate this using the natural neighbour interpolation
3) Both the discrete 3D Voronoi diagram, as well as it interpolated volume can be imported into GRASS for analysis and in MayaVi2 for visualization

And on the extra possibilities that `CreateDiscreteVD` offers, such as the modelling of generalized Voronoi diagrams, it can be concluded that the discrete 3D Voronoi diagram can be considered to be a well-suited, powerful tool for modelling 3D continuous data in geosciences.

# BIBLIOGRAPHY


Bailey, T. C. and A. C. Gatrell (1995). *Interactive spatial data analysis*, Harlow Longman Scientific & Technical.

Bak, P. R. G. and A. J. B. Mill (1989). "Three dimensional representation in a Geoscientific Resource Management System for the minerals industry." *Three dimensional applications in GIS*: 155-182.

Baxes, G. A. (1994). *Digital Image Processing: Principles and Applications*. New York, Wiley.

Behnel, S. and R. Bradshaw (2008). Cython**:** C extensions for Python.

Bentley, J. L. and J. H. Friedman (1979). "Data Structures for Range Searching." *ACM Comput. Surv.* **11**(4): 397-409.

Boissonnat, J. D. and F. Cazals (2001). "Natural neighbor coordinates of points on a surface." *Computational Geometry: Theory and Applications* **19**(2-3): 155-173.

Boissonnat, J. D. and F. Cazals (2002). "Smooth surface reconstruction via natural neighbour interpolation of distance functions." *Computational Geometry: Theory and Applications* **22**(1-3): 185-203.

Bresenham, J. (1977). "Linear algorithm for incremental digital display of circular arcs." *Communications of the ACM* **20**(2): 100-106.

Burrough, P. A. and R. McDonnel (2006). *Principles of Geographical Information Systems*. Oxford, Oxford University Press.

Carr, H., J. Snoeyink, et al. (2003). "Computing contour trees in all dimensions." *Computational Geometry: Theory and Applications* **24**(2): 75-94.

CGAL (2009). CGAL, the Computational Geometry Algorithms Library.

Courrioux, G., S. Nullans, et al. (2001). "3D volumetric modelling of Cadomian terranes (Northern Brittany, France): An automatic method using Voronoi diagrams." *Tectonophysics* **331**(1-2): 181-196.

de By, R. A. (2000). *Principles of Geographic Information Systems*. Enschede, International Institute for Aerospace Survey and Earth Sciences (ITC).

Delerue, J. F., E. Perrier, et al. (1999). "New algorithms in 3D image analysis and their application to the measurement of a spatialized pore size distribution in soils." *Physics and Chemistry of the Earth, Part A: Solid Earth and Geodesy* **24**(7): 639-644.

Descartes, R. (1644). *Le monde de Mr Descartes, ou Le Traité de la Lumiere*. Paris.

Devillers, O. and M. Teillaud (2003). *Perturbations and vertex removal in a 3D delaunay triangulation*. Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms.

Dirichlet, G. L. (1850). "Über die Reduction der positieven quadratischen Formen mit drei unbestimmten ganzen Zahlen." *Journal für die Reine und Angewandte Mathematik*(40): 209-227.

Dong, P. (2008). "Generating and updating multiplicatively weighted Voronoi diagrams for point, line and polygon features in GIS." *Computers & Geosciences* **34**(4): 411-421.

Drysdale, R. L. and D.-T. Lee (1978). *Generalized Voronoi diagrams in the plane*. Proceedings of the 16th Annual Allerton Conference on Communications, Control and Computing.

Dynamic Graphics Inc. (2008). "EarthVision." from http://www.dgi.com/earthvision/evmain.html.

ESRI (1998). ESRI Shapefile Technical Description - An ESRI White Paper.

ESRI. (2009). "ArcGIS Desktop Help 9.3 - Displaying netCDF data." from http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?id=3295&pid=3291&topicname=Displaying_netCDF_data.

Fuchida, T., M. Kashima, et al. (2005). "Constructing three-dimensional discrete Voronoi diagrams by the incremental method and application to self-organizing maps." *Systems and Computers in Japan* **36**(5): 55-67.
Gahegan, M. and I. Lee (2000). "Data structures and algorithms to support interactive spatial analysis using dynamic Voronoi diagrams." *Computers, Environment and Urban Systems* **24**(6): 509-537.
Gold, C. M. (1989). "Surface interpolation, spatial adjacency and GIS." *Three dimensional applications in GIS*: 21-35.
Gold, C. M. and G. Edwards (1992). "The Voronoi spatial model: two- and three-dimensional applications in image analysis." *ITC Journal* **1992-1**: 11-19.
Gourret, J. P. and J. Paille (1987). "IRREGULAR POLYGON FILL USING CONTOUR ENCODING." *Computer Graphics Forum* **6**(4): 317-325.
GRASS Development Team (2008). Geographic Resources Analysis Support System (GRASS) Software.
GRASS GIS. (2007, 2007-02-26). "GRASS GIS: v.surf.bspline." from http://grass.itc.it/grass63/manuals/html63_user/v.surf.bspline.html.
GRASS GIS (2008). "GRASS GIS: Documentation."
GRASS GIS. (2008). "NVIZ 3-D GRASS Interface." from http://grass.itc.it/nviz/index.html.
Guibas, L. J. and J. Stolfi (1983). *PRIMITIVES FOR THE MANIPULATION OF GENERAL SUBDIVISIONS AND THE COMPUTATION OF VORONOI DIAGRAMS*. Conference Proceedings of the Annual ACM Symposium on Theory of Computing.
Haralick, R. M., S. R. Sternberg, et al. (1987). "Image analysis using mathematical morphology." *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-9**(4): 532-550.
Hengl, T. (2006). "Finding the right pixel size." *Computers & Geosciences* **32**(9): 1283-1298.
Heywood, I., S. Cornelius, et al. (2006). *An Introduction to Geographical Information Systems*, Prentice Hall.
Hsieh, H.-H. and W.-K. Tai (2005). "A simple GPU-based approach for 3D Voronoi diagram construction and visualization." *Simulation Modelling Practice and Theory* **13**(8): 681-692.
Jamil, N., T. M. T. Sembok, et al. (2008). *Noise removal and enhancement of binary images using morphological operations*. Proceedings - International Symposium on Information Technology 2008, ITSim.
JewelSuite. (2009). "JewelSuite - Geological & Reservoir Simulation Software." from http://www.jewelsuite.com.
Ji, L., J. Piper, et al. (1989). "Erosion and dilation of binary images by arbitrary structuring elements using interval coding." *Pattern Recognition Letters* **9**(3): 201-209.
Karssenberg, D. and K. De Jong (2005). "Dynamic environmental modelling in GIS: 1. Modelling in three spatial dimensions." *International Journal of Geographical Information Science* **19**(5): 559-579.
Kaufman, A. E. (1996). "Volume visualization." *ACM Computing Surveys* **28**(1): 164-167.
Kirkpatrick, D. G. (1979). *Efficient computation of continuous skeletons*. proceedings of the 20th Annual I.E.E.E. Symposium on Foundations of Computer Science, Los Alamitos, CA, USA, Computer Society Press.
Langetepe, E. and G. Zachmann (2006). *Geometric data structures for computer graphics*, A K Peters, Ltd.
Ledoux, H. (2006). Modelling Three-dimensional Fields in Geoscience with the Voronoi Diagram and its Dual. *GIS Research Centre,School of Computing*. Glamorgan, University of Glamorgan/Prifysgol Morgannwg. **Doctor of Philosophy:** 168.
Ledoux, H. and C. Gold (2005). An Efficient Natural Neighbour Interpolation Algorithm for Geoscientific Modelling *Developments in Spatial Data Handling*. P. F. Fisher. Springer Berlin Heidelberg**:** pp. 97-108.

Lee, D.-T. and R. L. Drysdale (1981). "Generalization of Voronoi diagrams in the plane." *SIAM Journal of Computing* **10**: 73-87.
Li, C., J. Chen, et al. (1999). "A raster-based method for computing Voronoi diagrams of spatial objects using dynamic distance transformation." *International Journal of Geographical Information Science* **13**(3): 209-225.
Liu, Y., X. Chen, et al. (2009). "Semantic clustering for region-based image retrieval." *Journal of Visual Communication and Image Representation* **20**(2): 157-166.
Lorensen, W. E. and H. E. Cline (1987). "Marching cubes: a high resolution 3d surface construction algorithm." *Computer Graphics (ACM)* **21**(4): 163-169.
Majdandzic, I., C. Trefftz, et al. (2008). *Computation of voronoi diagrams using a graphics processing unit*. 2008 IEEE International Conference on Electro/Information Technology, IEEE EIT 2008 Conference.
Martini, H. and S. Wu (2009). "Geometric dilation of closed curves in normed planes." *Computational Geometry: Theory and Applications* **42**(4): 315-321.
Mead, R. (1967). "A mathematical model for the estimation of inter-plant competition." *Biometrics* **23**(2): 189-205.
Mitasova, H., L. Mitas, et al. (1995). "Modeling spatially and temporally distributed phenomena: New methods and tools for GRASS GIS." *International Journal of GIS* **9**(4): pp. 443-446.
Newman, T. S. and H. Yi (2006). "A survey of the marching cubes algorithm." *Computers and Graphics (Pergamon)* **30**(5): 854-879.
Okabe, A., B. Boots, et al. (1994). "Nearest neighbourhood operations with generalized Voronoi diagrams: a review." *International Journal of Geographical Information Science* **8**(1): 43 - 71.
Okabe, A., B. Boots, et al. (1992). Spatial Tessellations - Concepts and Applications of Voronoi Diagrams, John Wiley. **1st edition:** 671 pages.
Oliphant, T. O. (2006). *Guide to NumPy*, Trelgol Publishing, USA.
Park, S. W., L. Linsen, et al. (2006). "Discrete Sibson interpolation." *IEEE Transactions on Visualization and Computer Graphics* **12**(2): 243-253.
Perez, C. and L. Traversoni (1996). *Finite element simulation of shallow waters using natural neighbors techniques*. International Conference on Computational Methods in Water Resources, CMWR.
Petit, M., C. Claramunt, et al. (2008). A design process for the development of an interactive and adaptive GIS. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. **5373 LNCS:** 96-106.
Ramachandran, P. and G. Varoquaux (2008). Mayavi2.
Rigaux, P., M. O. Scholl, et al. (2002). *Spatial Databases - with application to GIS*. San Franscisco, Morgan Kaufmann.
Rong, G. and T.-S. Tan (2006). *Jump Flooding in GPU with Applications to Voronoi Diagram and Distance Transform*. ACM Symposium on Interactive 3D Graphics and Games, Redwood City, California, USA.
Rong, G. and T. S. Tan (2007). *Variants of jump flooding algorithm for computing discrete Voronoi diagrams*. Proceedings - ISVD 2007 The 4th International Symposium on Voronoi Diagrams in Science and Engineering 2007.
Sambridge, M., J. Braun, et al. (1995). "Geophysical parametrization and interpolation of irregular data using natural neighbours." *Geophysical Journal International* **122**(3): 837-857.
Samet, H. (1982). "Neighbor finding techniques for images represented by quadtrees." *Computer Graphics and Image Processing* **18**(1): 37-57.
Schlumberger (2009). "Petrel Seismic-to-Simulation Software."

Schueller, A. (2007). "A nearest neighbor sweep circle algorithm for computing discrete Voronoi tessellations." *Journal of Mathematical Analysis and Applications* **336**(2): 1018-1025.
Sharir, M. (1985). "Intersection and closest-pair problems for a set ofplanar discs." *SIAM Journal of Computing* **14**: pp. 448-468.
Sibson, R. (1980). "A vector identity for the Dirichlet tessellation." *Mathematical Proceedings of the Cambridge Philosophical Society* **87**(01): 151-155.
Sibson, R., Ed. (1981). *A brief description of natural neighbour interpolation*. Interpreting Multivariate Data. Chichester, USA, Wiley.
Spitzig, W. A., J. F. Kelly, et al. (1985). "Quantitative characterization of second-phase populations." *Metallography* **18**(3): 235-261.
Sugihara, K. and H. Inagaki (1995). "Why is the 3D Delaunay triangulation difficult to construct?" *Information Processing Letters* **54**: pp. 275-280.
Talbert, D. A. and D. Fisher (2000). *An empirical analysis of techniques for constructing and searching K-dimensional trees*. Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
Tang, S. and S. P. Chen (2008). *Data cleansing based on mathematic morphology*. 2nd International Conference on Bioinformatics and Biomedical Engineering, iCBBE 2008.
Tomlin, C. D. (1990). *Geographic Information Systems and cartographic modeling*. Englewood Cliffs, N.J. (USA), Prentice Hall.
Tsai, V. J. D. (1993). "Fast topological construction of delaunay triangulations and voronoi diagrams." *Computers and Geosciences* **19**(10): 1463-1474.
Tse, R. O. C. and C. Gold (2004). "TIN meets CAD--extending the TIN concept in GIS." *Future Generation Computer Systems* **20**(7): 1171-1184.
Tucker, G. E., S. T. Lancaster, et al. (2001). "An object-oriented framework for distributed hydrologic and geomorphic modeling using triangulated irregular networks." *Computers and Geosciences* **27**(8): 959-973.
van Bemmelen, J., W. Quak, et al. (1993). Vector vs. Raster-Based Algorithms for Cross Country Movement Planning *Auto-Carto 11*. Minneapolis, Minnesota**:** 304-317.
Van Gelder, A. and J. Wilhelms (1994). "Topological considerations in isosurface generation." *ACM Transactions on Graphics* **13**(4): 337-375.
van Oosterom, P. (1999). Spatial access methods. *Geographical Information Systems* P. A. Longley, M. F. Goodchild, D. J. Maguire and D. W. Rhind, John Wiley & Sons. **1:** pp. 385 - 400.
van Rossum, G. (2008). "Python Reference Manual."
Voronoi, G. (1908). "Nouvelles applications des paramel€tres continus al la thel orie des formes quadratiques - Premier mel moire: Sur quelques proprietl tel s des formes quadratiques positives parfaites." *J. Reine Angew. Math.* **133**: 242.
VTK. (2009). "VTK - The Visualization Toolkit." from http://www.vtk.org/.
Watson, D. F. (1992). *Contouring: a guide to the analysis and display of spatial data*. Oxford, United Kingdom, Pergamon Press.
Wigner, E. and F. Seitz (1933). "On the constitution of metallic sodium." *Physical Review* **43**(10): 804-810.
Wikipedia (2006). A visualization of results obtained by running the Python kd-tree-construction program. Kdtree_2d.svg. **370 × 368 pixels:** A visualization of results obtained by running the Python kd-tree-construction program on [(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)].
Winter, S. and A. U. Frank (2000). "Topology in Raster and Vector Representation." *GeoInformatica* **4**(1): 35-65.

Worboys, M. F. and M. Duckham (2004). *GIS: A computing perspective*, CRC Press.
Wu, H. S., R. Xu, et al. (2005). "Segmentation of intestinal gland images with iterative region growing." *Journal of Microscopy* **220**(3): 190-204.
Yanalak, M. (2003). "Effect of gridding method on digital terrain model profile data based on scattered data." *Journal of Computing in Civil Engineering* **17**(1): 58-67.
Yanalak, M. (2004). "Sibson (natural neighbour) and non-Sibsonian interpolation for digital elevation model (DEM)." *Survey Review* **37**(291): 360-376.
Yap, C. K. (1987). "An O(n log n) algorithm for the Voronoi diagram of a set of simple curve segments." *Discrete Computational Geometry* **2**: pp 365-393.
Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*. Austin, Texas, United States, Society for Industrial and Applied Mathematics.
Zhang, H. and C. Thurber (2005). "Adaptive mesh seismic tomography based on tetrahedral and Voronoi diagrams: Application to Parkfield, California." *Journal of Geophysical Research B: Solid Earth* **110**(4): 1-13.
Zhao, R., Z. Li, et al. (2008). A Hierarchical Raster Method for Computing Voronoi Diagrams Based on Quadtrees. *Computational Science — ICCS 2002***:** 1004-1013.

# APPENDIX I - INTERNAL STRUCTURE OF DATA FILES

The file formats used, and described in the previous section all have their own characteristics. The main characteristics of these files is described below.

## I.1. 3D POINT DATA SET OF SEEDS

The 3D point data set containing the seeds has the following internal structure:

n (number of points)

| | | | |
|---|---|---|---|
| $x_1$ | $y_1$ | $z_1$ | $a_1$ |
| $x_2$ | $y_2$ | $z_2$ | $a_2$ |
| .. | .. | .. | .. |
| $x_n$ | $y_n$ | $z_n$ | $a_n$ |

Where $x_i$ , $y_i$ and $a_i$ represent the x- and y-coordinates and the attribute value of point *i* respectively.

These files are not compressed, but since the data contains vector points only, and not raster information, the files tend to be manageable.

## I.2. ASCII 3D DATA FILE

The ASCII 3D data file stores 3D raster information, and can become very large, with relatively few points. The internal structure of the point is the following:

```
          ⎧ north:   floating point
          ⎪ south:   floating point
          ⎪ east:    floating point
          ⎪ west:    floating point
   Header ⎨ top:     floating point
          ⎪ bottom:  floating point
          ⎪ rows:    integer
          ⎪ cols:    integer
          ⎩ levels:  integer

          ⎧ i1,j1,k1  i2,j1,k1  i3,j1,k1  i4,j1,k1
          ⎪ i1,j2,k1  i2,j2,k1  i3,j2,k1  i4,j2,k1
Image Data⎨ i1,j3,k1  i2,j3,k1  i3,j3,k1  i4,j3,k1
          ⎪ i1,j1,k2  i2,j1,k2  i3,j1,k2  i4,j1,k2
          ⎩    …         …         …         …
```

The header information contains the extent of the data that is stored in the file, as well as the number of rows (i), colums (j) and levels (k). This allows for this file to be geo-referenced, but the geo-referencing itself is done in GRASS by creating a workspace: an environment which specifies certain characteristics, such as what coordinate system is used. The characteristics of the workspace, together with the coordinates of the ASCII 3D file provide geo-referenced data. There is no standard extension for this file, but in this project, all ASCII 3D files received the *.A3D extension.

## I.3. VTK AND VTK XML FILES

For the exporting of raw 3D array data to VTK XML there was already a library available, so there was no need to create one. VTK imports both formats, but GRASS exports VTK files, not VTK XML files. There is no possibility of geo-referencing these files, which is logical since MayaVi2 is not a GIS, but a visualization tool.